НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра автоматики та управління в технічних системах

«До захисту допущено»

Завідувач кафедри

_____ О.І. Ролік

«___»_____2019 р.

Дипломний проект

на здобуття ступеня бакалавра

з напряму підготовки 6.050201 «Системна інженерія»

на тему: «Веб-застосунок для збору та відображення інформації про товари в інтернет-магазині»

Виконав:

студент IV курсу, групи ІА-51

Сулейманов Артем Андрійович _____

Керівник:

Ст. викладач Март Б. А. _____

Рецензент:

Доцент, к.т.н. Волокита А. М. _____

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших

авторів без відповідних посилань.

Студент _____

Київ – 2019 рік

Пояснювальна записка

до дипломного проекту

на тему: «Веб-застосунок для збору та відображення інформації про товари в інтернет-магазині»

Київ – 2019 рік

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

ПЗ – Програмне забезпечення.

HTTP – HyperText Transfer Protocol.

AJAX – Asynchronous JavaScript and XML.

HTML – HyperText Markup Language

CSS – Cascading Style Sheets.

JS – JavaScript.

IDE – Integrated Development Environment.

JSON – JavaScript Object Notation.

SPA – Single page application.

FTP – File Transfer Protocol.

ADLs – Architecture definition languages.

URL – Uniform Resource Locator.

MVVM – Model-View-ViewModel.

ВСТУП

На сьогоднішній день, електронна комерція стала невід'ємною частиною буденного життя, як для звичайних користувачів, так и для компаній світового рівня. За даними компанії «eMarketer» за 2017 рік, обсяг продажу товарів через інтернет-магазини по всьому світу склав 2,304 трильйони доларів США, що складає 10,2% від світового обсягу роздрібної торгівлі[1]. Ці цифри майже на чверть перевищують відповідні показники минулого року. Це показує безумовний тренд на зростання популярності цифрових продаж. Такі тренди, звичайно, зумовлюють великий попит світового бізнесу на нові дослідження та розробки у сфері купівлі/продажу товарів через мережу Інтернет.

Інтернет-магазини – веб-сайти, що працюють за принципом реальних магазинів. Кожен потенційний покупець (відвідувач сайту) може шукати товари у загальному каталозі, вивчати вигляд та характеристики товарів, перш ніж прийняти рішення про придбання тієї чи іншої продукції.

Одна із головних очевидних переваг інтернет-магазинів перед звичайними – мала трата часу на переміщення між ними. Набагато швидше відкрити нове вікно браузера, ніж ходити з одного магазина в інший, шукаючи де є потрібний товар, та де його найдешевше придбати.

Наступним кроком розвитку цієї переваги стали сервісі, що не продають свою власну продукцію, а збирають інформацію з різних інтернет-магазинів (на англійський манер, їх називають «агрегатори інтернет-магазинів»). Вони надають змогу порівнювати ціни на один и той самий товар, що представлений у різних магазинах, одразу в одному місці (на одній сторінці). Цей підхід зручний, як для покупців, так і для продавців. Перші мають змогу швидше обрати найбільш привабливу пропозицію. Для других це додатковий засіб звернути на себе увагу та додаткове місце контакту із вже заінтересованими у покупці клієнтами.

Актуальність проблеми, що розглядається у проекті зумовлена тим, що велика частина даних сервісів має одні і ті ж самі недоліки. Серед них можна виділити наступні, що найчастіше зустрічаються:

– сервіс має інтуїтивно незрозумілий користувацький інтерфейс;

– сервіс має дуже громіздкий інтерфейс, та реалізовує можливості які занадто рідко використовуються, але ускладнюють користування сервісом;

– сервіс реалізовується як звичайний веб-сайт із статичними сторінками, а не як веб-застосунок, що дає меншу швидкість роботи та погіршує враження користувача.

Метою даного проекту стала розробка веб-застосунку, що мав би усі необхідні користувачу можливості та був би зручним у користуванні, маючи простий та зрозумілий користувацький інтерфейс. Також розроблюваний застосунок повинен мати можливість збирати дані із різних джерел (інтернет магазинів) у автоматичному режимі, та мати можливість для легкого додавання нових джерел даних із мінімальними зусиллями з боку розробника.

Завдання, що винесені на даний проект, наступні:

– огляд предметної області, для визначення критеріїв оцінки існуючих рішень, та вимог до майбутнього застосунку. Огляд подібних рішень на ринку, їх переваг та недоліків;

– визначення, на основі огляду предметної області та існуючих рішень, основних вимог до архітектури, функціоналу та дизайну застосунку;

– вибір технологій та засобів для розробки програмного забезпечення, які дозволять якнайкраще виконати поставлені вимоги;

– розробка архітектури та структур даних (класів, інтерфейсів, тощо) для майбутнього застосунку;

– розробка логіки та написання ПЗ для клієнтської та серверної частин застосунку;

– розробка дизайну та верстка шаблону застосунку.

Практичне значення отриманих результатів розробки полягає в отриманні простого та зручного сервісу з надання послуг клієнтам у виборі товарів із мережі Інтернет, та інтересу до запуску такого сервісу з боку компанії Costpair Canada ltd.

Даний бакалаврський проект складається із наступних розділів:

– вступ;

– основні розділи, де кожен розділ описує вирішення одного або кількох завдань, що були визначені раніше;

– кожен основний розділ має власні висновки, що коротко описують результати, що отримані у розділі;

– загальні висновки, що описують та підсумовують результати виконання проекту;

– список використаних джерел із 16 найменувань;

– один додаток, у якому міститься діаграма класів мовою UML, для класів, що розроблені для веб-застосунку;

– ілюстративний матеріал.

Ілюстративний матеріал проекту включає 4 документи формату А3, а саме:

– діаграма прецедентів мовою UML;

– структурна схема веб-застосунку;

– діаграма розгортання мовою UML;

– схема бази даних для веб-застосунку.

Загальний обсяг пояснювальної записки проекту складає 71 сторінку, без урахування ілюстративного матеріалу.

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Загальний огляд

Так як сервіси, що розглядаються у даному розділі – веб-сайти та веб-застосунки, необхідно розглядати їх окремо з точки зору користувача та з точки зору розробника/адміністратора.

До першої категорії слід віднести такі критерії оцінки як:

- спектр доступних можливостей;
- користувацький інтерфейс;
- швидкодія сервісу.

З точки зору розробника/адміністратора можна виділити наступні критерії оцінки:

- можливість та зручність додавання нових джерел даних;
- стійкість до навантажень;
- рівень кібербезпеки сервісу.

1.1.1 Для усіх сервісів, що об’єднують інформацію із різних інтернет-магазинів, можна виділити дві головні можливості, які вони повинні надавати користувачу.

По-перше, це об’єднання каталогів усіх магазинів в один. Такий єдиний каталог створює абстракцію, даючи можливість шукати товар одразу у великій кількості магазинів, так само, як користувач шукав би його в одному. Це стосується не тільки пошуку по назві, а й використання різноманітних фільтрів, для більш точного пошуку товарів із потрібними характеристиками.

Друга головна можливість це порівняння цін на один і той самий товар від

різних продавців, шляхом виведення їх поряд у список. Також у цей список включається коротка інформація про кожного продавця (назва, можливість та умови доставки товару, посилання на веб-сайт продавця, тощо). Це дозволяє клієнту швидко обрати найпривабливішу саме для нього пропозицію.

Крім цього мінімального набору, є ще кілька більш-менш популярних серед користувачів можливостей. Серед них варто виділити

- збереження посилань на товари у окремому списку користувача (це називають «бажаним», «обраним»);
- порівняння не тільки цін, а й усіх характеристик товарів одночасно, на окремій сторінці сервісу;
- Можливість оцінювати товари та/або коментувати кожен товар і, відповідно, бачити оцінки та коментарі інших користувачів.

Також, обов’язково слід зазначити, що, станом на 2018 рік, більше половини трафіку мережі Інтернет, а зокрема саме покупок/продажів припадає на мобільні телефони та планшети[1]. Отже, під час огляду, необхідно звертати увагу на оптимізацію та коректне відображення веб-сайтів при відкритті їх з мобільних пристроїв.

Обов’язково слід звертати увагу і на швидкість завантаження сторінок сервісу. Це важливо для користувача, бо занадто довге очікування погіршує враження про сервіс, а може й зовсім лишити бажання ним користуватися, особливо на мобільних пристроях. Також це важливо для пошукових систем, без яких сьогодні неможливий розвиток популярності сервісів для широкого кола користувачів. Варто звернути увагу, що для пошукових систем є особливо важливою швидкість завантаження першої сторінки на яку потрапляє користувач.

1.1.2 Розглядаючи методи отримання даних для сервісу, можна виділити два варіанти, самостійний збір даних (вручну або автоматизовано) та укладення

договорів з продавцями, щоб вони самостійно надавали інформацію про свою продукцію.

Перший метод підходить для нових та/або невеликих сервісів, що не мають достатньої репутації для співпраці із відомими продавцями, або не мають достатньо ресурсів для реалізації окремої частини сервісу для взаємодії із продавцями. Розробник сервісу самостійно (можливо, користуючись стороннім програмним забезпеченням) забирає інформацію від різних покупців. Тут варто приділити увагу правовій стороні питання, щоб вся інформація збиралася лише із відкритих джерел та методами, що дозволені відповідними країнами де зареєстровано сервіс. У цьому разі вся відповідальність за точність інформації, що надається клієнтам, лежить лише на власниках сервісу.

Другий метод навпаки підходить для сервісів, що вже мають репутацію, та з якими продавці самі виявляють бажання співпрацювати. Цей метод реалізовується через окреме API, що надає розробник сервісу. Власник сервісу та продавець укладають угоду, за якою останній самостійно віддає інформацію про продукцію в оговореному форматі даних. При використанні цього методу, ступінь відповідальності за точність інформації, що надається клієнтам, розподіляється між обома сторонами угоди у частках, що визначені самою угодою.

1.2 Огляд існуючих рішень

Перш за все, треба зазначити, що всі сервіси, що розглядаються, є комерційними продуктами та створені як власницьке програмне забезпечення. Тому, у цьому проекті вони будуть розглядатися лише з клієнтської точки зору.

Оскільки, компанія Costpair Canada ltd., що має інтерес до даного проекту націлена першочергово на ринок Канади, то особлива увага у розділі буде присвячена сервісам канадського ринку.

1.2.1 Перший сервіс, який буде оглянутий – shopbot.ca. Приклад інтерфейсу зображено на рисунку 1.1.

Даний сервіс працює виключно у Канаді, та має основну напрямленість на електронні пристрої. Інші категорії товарів у ньому не представлені, або представлені дуже малою кількістю товарів. Головний його недолік полягає у користувацькому інтерфейсі, дизайн якого зроблений у однотонній кольоровій гамі, що ускладнює зоровий пошук потрібних користувачу елементів управління.

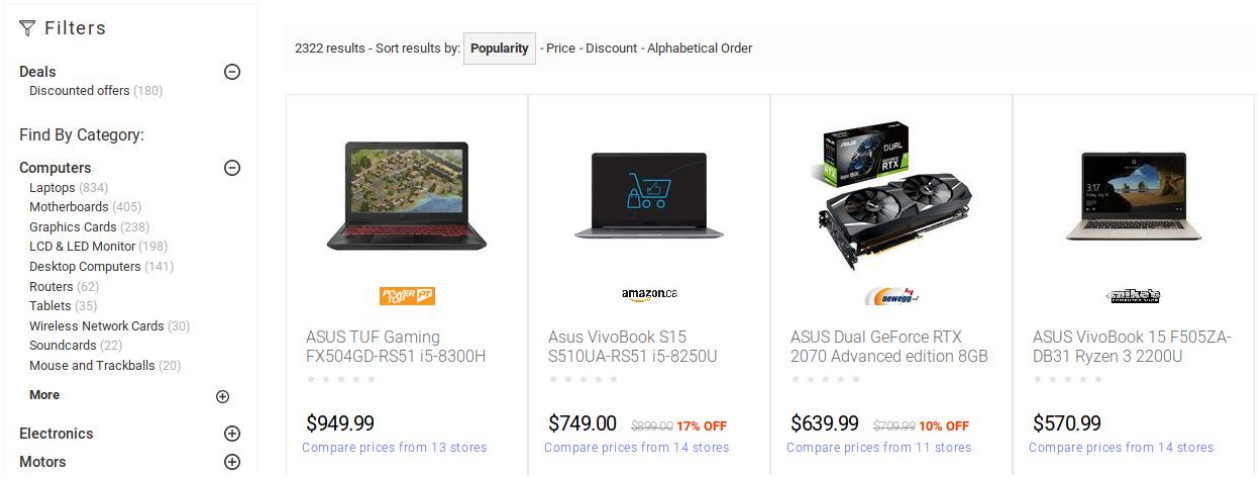

Рисунок 1.1 – Скріншот сайту shopbot.ca

Переваги даного сервісу:

– велика та актуальна база товарів, що часто оновлюється;
– зручний дизайн карток товарів.

Недоліки:

– непродуманий дизайн користувацького інтерфейсу, де сірі написи зливаються, що ускладнює пошук потрібної функції;
– скупий функціонал, що зводиться тільки до порівняння цін;

– не інтуїтивний дизайн сервісу для мобільних пристроїв;

– направленість сервісу лише на електронні прилади.

1.2.2 Наступний сервіс для огляду – hotline.ua. Приклад інтерфейсу зображено на рисунку 1.2.

Даний сервіс є одним із найдавніших на ринку світу (починався ще як паперове видання), проте він має орієнтацію лише на український ринок. Представлений із широким категорій товарів. Слід відзначити дуже продуманий користувацький інтерфейс, що компенсує надмірне нагромадження функцій сервісу. Користувачі можуть реєструватися у сервісі, що дозволяє оцінювати, обмінюватися думками про різні товари та створювати власні списки для зберігання інформації про різні групи товарів. Також має, як гарну мобільну версію, так і власні додатки для популярних мобільних операційних систем.

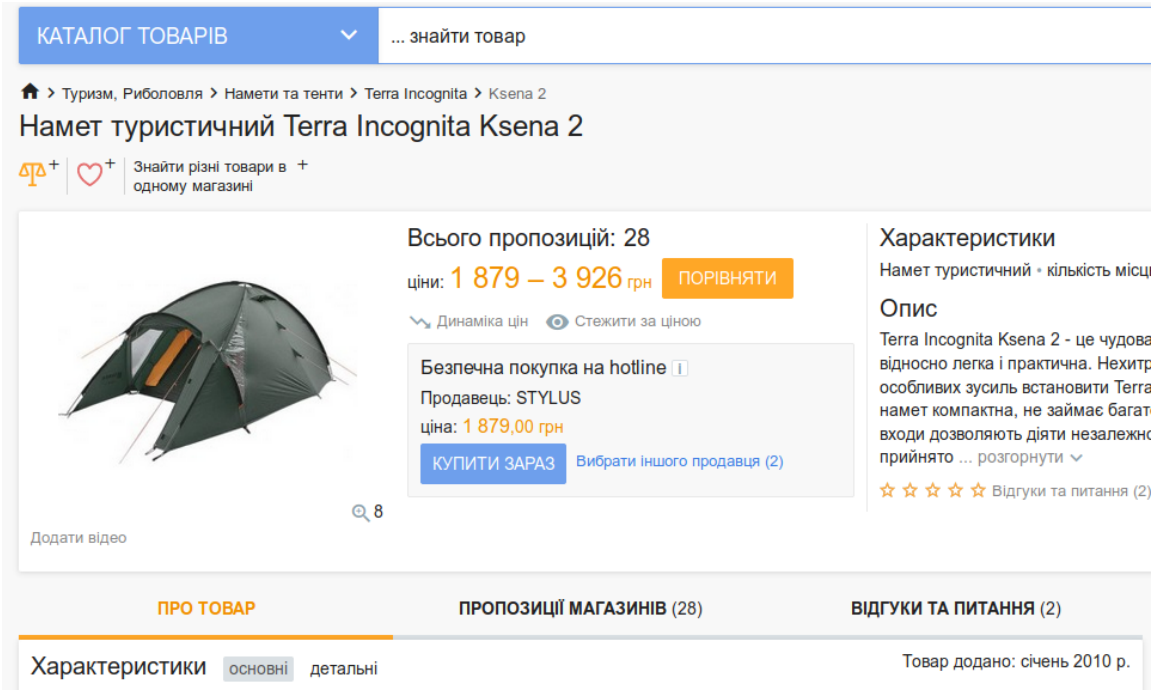

Рисунок 1.2 – Скріншот сайту hotline.ua

Переваги даного сервісу:

– Кольорова гама із гарним контрастом, на якому добре видно елементи управління;

– зручний дизайн сторінки порівняння характеристик товарів;

– зручний та інтуїтивно зрозумілий інтерфейс при відкритті з мобільних пристроїв;

– наявність товарів із усіх сфер життя.

Недоліки:

– нагромадження елементів управління, що заплутує користувача, та ускладнює користування сервісом.

1.2.3 Останній з оглянутих сервісів – pricebat.ca. Приклад інтерфейсу зображено на рисунку 1.3.

Ще один сервіс канадського походження. Як і попередній розглянутий сервіс, має давню історію, проте, на відміну від hotline.ua, розвивається дуже повільно, має застарілі, як користувацький інтерфейс, так і програмну базу.

Суттєвих переваг перед подібними сервісами не має.

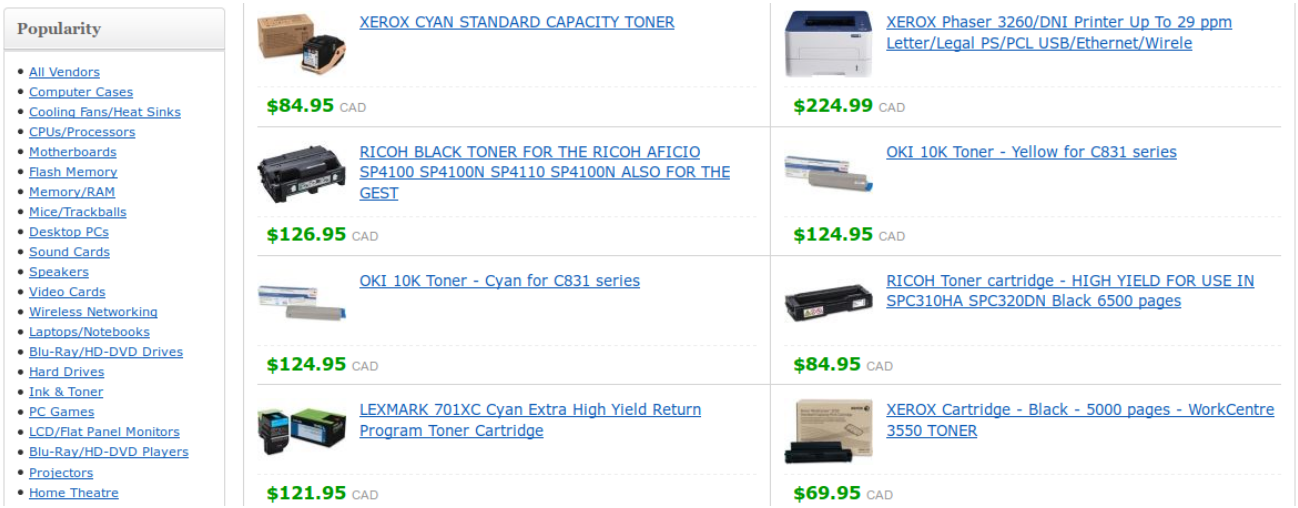

Рисунок 1.3 – Скріншот сайту pricebat.ca

Недоліки:

- застарілий та незручний користувацький інтерфейс;
- відсутність інтерактивності (AJAX);
- використання застарілих технологій, на кшталт, Adobe Flash Player, що робить роботу веб-сайту повільною та наражає користувачів веб-сайту на небезпеку ураження свого комп’ютера шкідливим програмним забезпеченням.

Висновки до розділу 1

У даному розділі було сформовано список критеріїв для оцінки сервісів, що збирають та відображають інформацію з різних інтернет магазинів. За цими критеріями було оцінено кілька прикладів даних сервісів. Визначені, як їх переваги, так і недоліки. Результати цих оглядів, а також визначенні критерії оцінки допоможуть у наступному розділі сформувати вимоги до веб-застосунку, що розробляється у рамках даного проекту.

2 ВИЗНАЧЕННЯ ВИМОГ ДО РОЗРОБЛЮВАНОГО ВЕБ-ЗАСТОСУНКУ

У попередньому розділі було сформовано бачення предметної області та розглянуто кілька веб-застосунків, загальна ідея яких збігається із ідеями, що закладені у розробку даного проекту.

Як можна бачити з попереднього розділу, різні рішення мають свої переваги та недоліки. Отже, варто брати приклад із найбільш вдалих рішень, та уникати помилок із невдалих рішень.

На основі зробленого огляду існуючих рішень, та враховуючи винесені на проект завдання, були визначені наступні основні вимоги до майбутнього веб-застосунку.

2.1 Функціональні можливості

Веб-застосунок, що розробляється у рамках даного проекту, повинен надавати користувачу наступні можливості (функціонал):

- перегляд, на окремій сторінці, списку товарів із обраної користувачем категорії;
- фільтрація списку товарів, за наданими користувачу фільтрами;
- пошук товарів за назвою;
- власна сторінка для кожного товару, із відображенням детального списку характеристик даного товару;
- відображення, на сторінці товару, списку усіх продавців, які пропонують даний товар. У списку має бути додаткова інформація про товар та продавця, така як: назва, умови доставки товару, коротка інформація про товар від продавця, посилання на веб-сайт продавця, тощо;
- створення, зберігання та перегляд «бажаних» користувачем товарів у вигляді списку;

- додавання та видалення товарів із списку «бажаного»;
- очищення одразу усього списку «бажаного»;
- перегляд списку «бажаних» товарів на окремій сторінці;
- створення, зберігання та перегляд списків порівняння за обраною категорією;
- додавання та видалення товарів із кожного списку порівняння;
- очищення одразу усього списку порівняння для обраної категорії;
- можливість оцінювати кожен товар один раз, оцінкою від 1 до 5 (із одиничним кроком);
- перегляд узагальненої середньої оцінки товару від усіх користувачів (від 1 до 5, із кроком 0,5);
- перегляд діапазону цін на товар у форматі «мінімальна ціна – максимальна ціна»;
- відображення середньої (медіанної) ціни на товар;
- відображення акційної ціні на товар, якщо у якогось із продавців даного товару діє акція на нього;
- можливість порівняння характеристик товарів із одного списку порівняння на окремій сторінці;

Для кращого розуміння майбутнього функціоналу, було розроблено діаграму прецедентів мовою UML. Вона міститься у ілюстративних матеріалах проекту.

2.2 Користувацький інтерфейс

Наступний перелік вимог відноситься до проектування та реалізації зовнішнього вигляду веб-застосунку із точки зору користувача:

- кольорова гама користувацького інтерфейсу повинна мати 2-4 основних кольори, що добре контрастують між собою, та до чотирьох відтінків кожного із цих кольорів;

- усі функціонально схожі елементі управління (кнопки, списки, тощо) повинні бути виконані в одному дизайні та мати схожі ефекти анімацій (при наведенні миші, вибору елемента списку, тощо). Допускаються незначні відмінності для виділення більш конкретного функціоналу (кнопки для навігації та кнопки для підтвердження дії, тощо);
- елементи управління не повинні робити інтерфейс занадто громіздким. Тому повинні бути розташовані не більше трьох різнотипних елементів в одній візуальній групі. Елементі розташовані поряд повинні мати схожі розміри;
- перша, за замовчуванням, сторінка веб-застосунку має включати елемент для навігації по дереву категорій товарів, посилання на інформацію щодо сервісу та компанії-власника сервісу, списки найбільш популярних товарів серед інших користувачів сервісу;
- уся користувацька частина застосунку повинна бути адаптована та оптимізована для коректного відображення на мобільних пристроях.

2.3 Архітектурні вимоги

Вимоги до загальної структури проекту наступні:

- реалізація каталогу товарів за категоріями, що утворюють ієрархічне дерево;
- можливість написання програмних скриптів для додавання нових джерел даних;
- веб-застосунок повинен відповідати сучасним вимогам пошукових систем, для коректної індексації та відображення у результатах пошуку;
- зберігання списку обраних товарів користувача у локальному сховищі даних користувацького браузеру;
- зберігання списків порівняння для різних категорій у локальному сховищі даних користувацького браузеру;

– оскільки веб-застосунок має зберігати дуже великий обсяг інформації, і інформація, що надається користувачам, пов’язана із фінансами, то весь обсяг даних веб-застосунку має зберігатися у надійному сховищі, що забезпечує цілісність даних.

Варто зазначити, що у минулому розділі було описано два різні можливі методи сервісу до отримання даних. У даному проекті було обрано другий з описаних методів, із самостійним автоматизованим збором даних, через те, що проект новий, а отже є невідомим на ринку. Тому у вимоги була винесена можливість написання програмних скриптів для отримання даних. У межах даного проекту планується реалізація одного або кількох таких скриптів для прикладу.

Висновки до розділу 2

У даному розділі були чітко визначені вимоги до проектування, розробки та кінцевого вигляду веб-застосунку, який є метою цього проекту. Ці вимоги достатньо описують архітектуру, функціонал та користувацький інтерфейс майбутнього веб-застосунку. Надалі це буде слугувати «дорожньою мапою» розробки веб-застосунку та допоможе точно обрати найбільш відповідні програмні інструменти, архітектурні підходи та алгоритми.

3 ВИБІР ТЕХНОЛОГІЙ ТА ЗАСОБІВ ДЛЯ РОЗРОБКИ

3.1 Мова програмування JavaScript

JavaScript – мова програмування високого рівня, яка є однією із найпопулярніших у сфері розробки програмного забезпечення. Вона є реалізацією стандарту ECMAScript (ECMAScript 2018 – остання версія стандарту, на момент написання даного розділу). Крім JavaScript існують і інші реалізації ECMAScript, наприклад ActionScript, JScript та QtScript. Проте вони не досягли такого успіху та майже всі визнані застарілими.

JavaScript створювалася як мова, що дозволяє додавати інтерактивність до сторінок веб-сайтів (анімації, динамічної зміни змісту, тощо). Із часом, ця мова набирала популярність і тепер є основною мовою клієнтської веб-розробки, що підтримується усіма сучасними браузерами. Історично, програми мовою JavaScript називаються «скрипти».

JavaScript є інтерпретуємою, повною за Тюрінгом та має Сі-подібний синтаксис. Це об’єктно-орієнтована мова, що використовує прототипи. До ECMAScript 6 у JavaScript було відсутнє поняття класу основи об’єктно-орієнтованої парадигми. Проте, навіть з їх появою, JavaScript залишається прототипно-орієнтованою, а класи є лише «синтаксичним цукром» для роботи із прототипами. У JavaScript передбачено автоматичне керування пам’яттю та емуляція комп’ютера із нескінченною пам’яттю за допомогою системи «збирання сміття».

Так як JavaScript, у першу чергу, є мовою для браузерів, кожен із них має самостійно реалізувати усе необхідне для виконання скриптів JavaScript. Сюди можна віднести:

– ядро мови. Це частина, що описується стандартом ECMAScript. Сюди належать інтерпретатор, стандартні функції, система типів, оператори, тощо;

– document object model (DOM). Цей функціонал відсутній у ECMAScript, а описується стандартами, що розробляє організація W3C. Сюди належать можливості взаємодії JavaScript із сторінками веб-сайтів, що представлені у вигляді розмітки HTML;

– browser object model (BOM). Цей функціонал також стандартизується W3C, замість ECMAScript. Сюди належить периферійний функціонал взаємодії браузера із користувачем. Це геолокація, робота із cookies та локальним сховищем даних браузеру, асинхронні запити (AJAX), керування історією перегляду браузера, тощо.

Як можна бачити, JavaScript великою мірою залежить від середовища у якому вона виконується. У зв'язку з цим, варто також зазначити головний недолік даного підходу. Із віходом нової версії стандарту ECMAScript або W3C, розробник не може образу користуватися усіма новими можливостями, а повинен чекати, доки основні середовища виконання (браузери) оновлять свої рушії. Рушій – термін, що прийшов на зміну інтерпретатору, коли веб-застосунки стали занадто складні для простої інтерпретації, і розробники серед управління для JavaScript почали використовувати більш складні технології для оптимізації виконання коду.

Оновлення рушіїв для підтримки нових версій стандартну не є швидким процесом и може займати роки. Для вирішення цієї проблеми, було розроблено декілька різних, які компілюють код, написаний новою версією JavaScript, у більш давню версію мови. Таким чином, розробники можуть використовувати можливості, що вже реалізовані у нових стандартах, але ще не реалізовані популярними рушіями JavaScript. Такі програми називаються «транспілятори».

Мова JavaScript виділяється своєю простою системою типів. Вона складається із трьох примітивів (число, булеве значення, рядок) і трьох типів об'єктів (об'єкт, масив, функція). Типізація мови характеризується як динамічна (тип змінних визначається у процесі виконання програми), слабка (змінні можуть неявну приводитися до різних типів, навіть із втратою інформації),

неявна (змінним не потрібно вказувати тип при створенні, цим займається інтерпретатор).

Історично склалося, що програмне забезпечення, що розроблене мовою JavaScript має завжди модульну архітектуру. «Модуль» у даному розумінні це частина логіки, яка логічно структурована та винесена в окремий файл коду. При підключенні одного модулю (імпорту модулю) у коді іншого, модуль, що імпортує, отримує можливість використовувати ті дані, які явно віддаються (експортуються) у підключеному модулі. За класичним визначенням шаблонів проектування, можна це описати, як шаблон проектування «фасад»[2]. Окремий модуль може включати будь-який об’єм логіки та повинен експортувати один або більше об’єктів. Об’єктами, у даному випадку, можуть виступати змінні, константи чи функції. До недавнього часу у екосистемі JavaScript не було єдиної архітектури використання модулів. Це призвело до появи різних рішень, що можуть бути, як сумісні між собою, так і взаємовиключні. Порівняно нещодавно, у стандарті мови JavaScript, з’явилася концепція модулів, яка у майбутньому має бути реалізована у рушіях серед управління JavaScript та має витіснити усі рішення, що не сумісні із стандартом.

3.2 Мова програмування TypeScript

TypeScript – проміжна мова програмування, що компілюється у мову JavaScript. Вона була створена, щоб розширити мову JavaScript, додавши до неї можливість статичної, сильної, явної типізації.

TypeScript розробляється компанією Microsoft, та за своїми можливостями нагадує іншу мову програмування цієї компанії – C#.

За інформацією з офіційної документації, основними перевагами, які надає TypeScript, є[3]:

– повна зворотня сумісність із JavaScript. Скрипт, що написано на JavaScript

є також і скриптом на TypeScript;

– значне зменшення кількості потенційних помилок при розробці програмного забезпечення, завдяки можливостям статичної типізації;

– хоча статична типізація є обов’язковою частиною TypeScript, є можливість для розробника самому обирати, чи буде використовуватися сильна або слабка типізація, та чи буде використовуватися явна або неявна типізація;

– TypeScript підтримує всі типи, що є у стандарті ECMAScript;

– TypeScript надає можливість створення власних типів, на основі об’єднання або обмеження вже існуючих;

– концепція інтерфейсів надає також переваги під час розробки. Наприклад можливість автоматичного доповнення коду у IDE (спеціальні програми для оптимізації та прискорення процесу розробки);

– TypeScript дозволяє використовувати концепцію інкапсуляції із термінології об'єктно-орієнтованого програмування. Вона реалізує функціонал модифікаторів доступу, які не підтримуються у JavaScript (всі поля тут публічні, для створення приватних змінних використовуються контракти. Наприклад, іменування приватних змінних починається із символу нижнього підкреслення «_»). Модифікатори доступу дозволяють робити поля не тільки публічними (public, доступно із будь-якого місця коду), а ще й приватними (private, доступно лише у структурі де поле оголошено) та захищеними (protected, доступно лише у структурі де поле оголошено та у її нащадках). Також TypeScript дозволяє помічати поля структур як «тільки для читання» (readonly), що дозволяє робити константними поля структур;

– підтримка узагальненого програмування дозволяє описувати дії над певними типами даних, не конкретизуючи які саме типи будуть використовуватися. Це реалізує концепцію поліморфізму із термінології об'єктно-орієнтованого програмування;

– крім усього переліченого, TypeScript також є транспілятором. Він підтримує синтаксис останніх версій стандарту ECMAScript та дозволяє обрати якій версії стандарту повинен відповідати написаний код після компіляції;

Крім написання коду, TypeScript надає можливість створення фалів декларацій (або «об’яв»). У них описуються лише типи та інтерфейси. Файли декларацій можуть бути загальними або у вигляді модулів. У першому випадку, типи, що описані у файлі будуть доступні для використання у всьому проекті, у каталозі якого створено файл. При створенні файлу декларацій у вигляді модулю, для використання відповідних типів необхідно імпортувати файл декларацій у файл із кодом як звичайний модуль. Ця можливість використовується для опису типів бібліотек або модулів, які написані мовою JavaScript та не мають власного опису типів. У комплекті із компілятором TypeScript, також поставляються файли декларацій для стандартних функцій мови JavaScript.

Компілятор та супутні можливості TypeScript встановлюються у систему окремою програмою. У ній присутній зручний інтерфейс командного рядка для запуску компіляції із різними параметрами. Також усі параметри запуску команди компілятора и додаткові параметри роботи TypeScript можна задати описом у синтаксисі JSON у спеціальному файлі у кореневому каталозі.

3.3 Веб-фреймворк

Варто визначити різницю між веб-сайтом та веб-застосунком. Стаття від IBM, каже наступне[4]. Хоча, веб-застосунок є також і веб-сайтом, під першим терміном, зазвичай, розуміють ресурс у мережі Інтернет, що несе інформаційну ціль, простій у побудові, без динамічної зміни змісту, тощо. Веб-застосунки є наступним кроком до кращої і простішої взаємодії із користувачем, вони

інтерактивні, активно взаємодіють із користувачем, спонукають його до дій (підписки, покупки, поглибленого вивчення матеріалу, тощо).

Сучасні веб-застосунки у мережі Інтернет все більше нагадують звичайні комп'ютерні застосунки. Але їх перевага полягає у повній незалежності від операційної системи та складових деталей комп'ютера користувача. Абстракція, яку забезпечують браузери, дозволяє набір технологій (HTML+CSS+JS), які точно будуть підтримуватися клієнтським пристроєм.

Для розробки динамічних, швидких та сучасних веб-застосунків вже недостатньо простих анімацій, створених за допомогою JavaScript. Велика кількість архітектурних рішень та необхідного функціонала є спільна у більшості веб-застосунків, тому для їх розробки застосовують веб-фреймворки.

Веб-фреймворки – це «каркас» архітектури веб-застосунку. Це бібліотека готового коду, яка полегшує розробку за рахунок готових функцій для початку розробки проекту та для автоматичного виконання часто зустрічних «механічних» дій розробника.

На момент написання даного розділу існує три найбільш розвинуті веб-фреймворки для розробки веб-застосунків:

– AngularJs – найбільший за можливостями фреймворк, що розробляється та підтримується компанією Google. Він дотримується на клієнтському боці застусунку шаблону MVC (Model-View-Controller), що дає змогу перенести більшу частину розрахунків із серверу до клієнту. Це значно пришвидшує роботу складних застосунків. Проте, цей фреймворк є складним для вивчення, та містить багато коду, що зумовлює велику потребу у пам'яті, навіть для невеликих застосунків;

– ReactJs – легкий, не вимогливий до об'єму пам'яті фреймворк, що робить ухил у швидкість роботи застосунків. На відміну від AngularJs, він не розрахований на складні розрахунки, та призначений лише для спрощення створення інтерактивного та динамічного клієнтського інтерфейсу.

Розроблюється та підтримується компанією Facebook.;

– VueJs – фреймворк, що по своїй архітектурі та ідеології дуже схожий на ReactJs, та є його головним конкурентом. Створений нещодавно, він має ряд суттєвих переваг у швидкості над своїм конкурентом, за рахунок використання новіших браузерних технологій та можливостей. Із недоліків, що пов'язані із малим часом існування, має набагато меншу кількість сторонніх бібліотек та прикладів використання ніж ReactJs.

Для даного проекту, враховуючи наявність вимог до інтерактивності користувацького інтерфейсу та відсутність складних розрахунків на клієнтському боці, було вирішено використовувати фреймворк VueJs, бо він має найбільшу швидкодію, найменші вимоги до об'єму пам'яті та найлегших в освоєнні серед трьох оглянутих.

Як головна бібліотека коду VueJs, так і усі офіційні модулі до нього від його розробників повністю написані мовою JavaScript, та поставляються із файлами декларацій TypeScript, що дозволить легко використовувати цю мову для розробки веб-застосунку.

Головною ідеєю VueJs є використання «реактивності». У термінології розробки програмного забезпечення реактивність це концепція програмування, що спирається на розповсюдження асинхронних потоків даних. Потік, у даному випадку, це послідовне спрацювання функцій, як відповідей на події, що у свою чергу, ініціюють наступні події. Головна перевага даної концепції полягає у тому, що одні і ті ж дані, які знаходяться у різних частинах коду застосунку, завжди будуть однакові. І зміна їх в одному місці веде за собою лавинну зміну значень усіх змінних, що від них залежать.

Для створення елементів відображення та управління клієнтського інтерфейсу, за допомогою VueJs, використовуються «компоненти». Компоненти це об'єкти, що створюються розробником. Компоненти є основними «блоками» для побудови клієнтської частини застосунку.

Компонент, що оголошений один раз, можна створювати у нескінченній кількості екземплярів. Кожен екземпляр буде мати свій власний стан даних. Таким чином, компоненти є головною функціональністю для повторного використання написаного коду у VueJs.

Окрім компонентів, для повторного використання коду VueJs має функціональність домішок. Домішки є компонентами, які не використовуються самостійно, а можуть об’єднуватися із іншими компонентами поєднуючи функціональність домішки й компоненту.

Ось ще кілька переваг, що надає VueJs для розробника:

- містить вбудовані можливості для кешування даних, що значно прискорює та спрощує розробку застосунків на основі даного фреймворку;
- має офіційну бібліотеку, що інтегрується із інструментами розробника для браузерів та дозволяє відстежувати стан даних та стан дерева компонент прямо із вікна застосунку у браузері;

VueJs призначений, у першу чергу, для створення односторінкових застосунків (SPA). Це означає, що при першому запиті клієнта до сервера формується відповідь, у якій є всі необхідні файли для відображення усіх сторінок застосунку. А додаткові файли та дані, що необхідні лише для деякого функціоналу, завантажуються окремо, за необхідністю, за допомогою технології асинхронних запитів JavaScript (AJAX). Це значно пришвидшує роботу застосунку, і єдиним вузьким місцем залишається час завантаження першої сторінки.

VueJs, окрім основної бібліотеки коду фреймворку включає у себе цілу екосистему різних модулів, які інтегруються із фреймворком, та дозволяють вирішувати типові проблеми розробки складних веб-застосунків.

3.3.1 Система маршрутизації

Розробники VueJs надають готове рішення для створення та налаштування маршрутизації у екосистемі веб-застосунку, що використовує VueJs, за допомогою бібліотеки VueRouter. Ось головні переваги даного рішення:

- має глибокий зв’язок із основною бібліотекою коду фреймоврку, що зумовлює легку інтеграцію рішення маршрутизації у проект;
- має гнучкі можливості налаштування маршрутів (обов’язкові та необов’язкові сегменти маршруту, валідація сегментів, тощо) за допомогою регулярний виразів;
- підтримує різні режими роботи маршрутизації для гнучкого налаштування та сумісності із старими браузерами;
- має вбудовані можливості для роботи із параметрами маршрутів;
- містить готові рішення для інтерактивності. Анімації переходів між сторінками, налаштування зовнішнього вигляду користувацьких елементів управління для маршрутизації.

3.3.2 Сховище даних для фреймворку

Однією з основних складностей розробки сучасних складних веб-застосунків є проблема зберігання даних. Система реактивності, яку мають більшість сучасних фреймворків полегшує рішення цієї проблеми, проте не завжди, під час роботи застосунку, вдається легко синхронізувати дані, що отримані у різний час із різних джерел. Також проблемою є відсутність механізму зручною передачі даних у віддалені частини логіки застосунку.

Ці проблеми надихнули розробників на створення архітектурного шаблону

під назвою Flux, що являє собою глобальний стан даних усього застосунку. Дана архітектура була запропонована розробниками із компанії Facebook Очевидно, що таке сховище даних повинно бути єдине для усього застосунку, тому для його реалізації використовують шаблон «одинак». За описом на офіційному сайті, типова архітектура Flux має три основні типа сутностей[5]:

– сховище. Це є власне місце, де зберігаються самі дані. Стан даних сховища слугує першим джерелом істини для застосунку, і коли стан даних у будь-якому місці застосунку не відповідає стану даних у сховищі – дані застосунку вважаються невірними і відбраковуються. Застосунок не має права працювати напряму зі сховищем, із міркувань безпеки та цілісності даних;

– представлення. Це функціонал, що забезпечує отримання застосунком даних із сховища. Тут дані можуть зазнати певних змін, щоб бути віддані компоненту застосунку у тому вигляді, у якому компоненту буде зручно їх використовувати.

– дії. Дії є єдиною можливістю зміни даних у сховищі. Вони вони викликаються не напряму, а ініціюються за допомогою спеціального «диспетчеру». Диспетчер слідкує за порядком виклику дій, та не дозволяє виконуватись більш ніж одній дії одночасно. Таким чином за одну дію може бути змінена велика кількість даних у сховищі, за допомогою механізму підписки, та зберігається цілісність даних. Дії можуть, як просто ініціювати певні запити та зміни даних, так і містити у собі певні дані (передані до них при виклику диспетчеру із застосунку), які будуть передані у сховище.

Як можна бачити, Flux є архітектурою, що забезпечує цілісність та послідовний доступ до даних, які є єдиними та істинними для усього застосунку.

Екосистема VueJs має своє власне рішення для реалізації архітектури Flux, яке легко інтегрується у застосунок, що розроблюється за допомогою даного фреймворку. Це рішення є бібліотекою коду під назвою Vuex.

Vuex є типовим представником Flux архітектури, що розширює її функціонал. У Vuex є окреме сховище, що зберігає дані. Для доступу до цих даних у незмінному, можна звернутися напряму об’єкту, що зберігає стан даних сховища. Для розробника це виглядає як звернення до сховища напряму, проте Vuex забезпечує обмеження, стежачи за тим, щоб при таких зверненнях дані можна було лише читати, а не змінювати.

При застосуванні із VueJs , Vuex автоматично інтегрується із компонентами, що створені розробником, для легкої та швидкої роботи із даними. Розробник має можливість ініціювати дії через диспетчер Vuex, що може бути викликаний із будь-якого місця коду компонентів.

3.3.3 Рендеринг веб-застосунку на сервері

Серверний рендеринг – прогресивний підхід до проектування веб-застосунків, при якому, при першому запиті до сервера, повністю формується сторінка, вже із завантаженими даними, яка буде відправлена у відповідь клієнту[6]. Це дозволяє вирішити, вже згадану вище, проблему односторінкових застосунків, пов’язану із швидкістю відповіді на перший запит. Так, як весь зміст сторінки повністю завантажується та формується на сервері, браузер приймає вже готову сторінку, що виглядає для нього статичною, та код мовою JavaScript, для додавання інтерактивності до застосунку.

VueJs надає готове рішення для серверного рендерингу із бібліотекою VueServerRenderer. Головне архітектурне рішення даної бібліотеки полягаю у тому, що у проекті формуються дві точки входу (перші файли, що запускають функціонал усього застосунку).

Перша з них – серверна. Цей код буде запущено на серверному обладнані при отриманні першого запиту від клієнту. Одразу буде завантажено усі необхідні дані для відображення сторінки, сформовано відповідну HTML

розмітку зі стилями, і відправлено повністю готову сторінку.

Друга із точок входу – клієнтська. Це точка входу коду, який буде передано клієнту, разом із готовою сторінкою, у відповідь на перший запит. Цей код містить необхідні інструкції для формування розміток усіх інших сторінок, та адреси віддалених ресурсів, де зберігаються дані сервісу.

При отриманні клієнтським браузером першої сторінки, клієнтський код запускаю процес «гідратації». Офіційна документація описує, що гідратацією називається процес додавання до статичної сторінки інтерактивності веб-застосунку[7]. Це включає у себе:

- створення та додавання оброблювачів подій на потрібні елементи управління користувацького інтерфейсу;
- запуск системи маршрутизації, якщо вона підключена до проекту;
- створення Vuex сховища (якщо ця бібліотека підключена до проекту) та заповнення його даними, що були отримані із відповіді серверу.

3.4 Збірка модулів

У зв'язку зі зростанням складності сучасних веб-застосунків неможливо уявити їх розробку без застосування спеціалізованих програм для збірки модулів. Вони призначені для того, щоб об’єднати великий набір файлів розробника у невелику кількість оптимізованих файлів, що будуть віддаватися у відповідь на запити клієнтів

Для даного проекту було вирішено використовувати програму збірки модулів Webpack. За інформацією із офіційної документації він працює наступним чином[8]. Отримуючи файл, що буде точкою входу для коду застосунку, Webpack рекурсивно об’єднує його із усіма його залежностями (під «залежностями» слід розуміти модулі, що імпортуються у файлі). На виході

розробник отримує один, або декілька послідовних файлів, які містять увесь необхідний програмний код проекту. Такі файли називаються «бандл», від англійського слова bundle, що означає «зв’язка». Webpack також надає можливість слідкувати за файлами у режимі реального часу, змінюючи бандл при кожній зміні початкових файлів. Також, Webpack може виконувати наступні функції

- інтеграція із сторонніми програмами дозволяє робити додаткові кроки у процесі створення збірки. Наприклад Webpack може отримувати файли мовою TypeScript та, використовуючи окремо встановлений компілятор даної мови, компілювати файли у JavaScript під час створення збірки;
- Webpack, у процесі створення збірки, автоматично знаходить та видаляє частини коду, що точно не буде використовуватися («мертвий» або недосяжний код), та оптимізує код, що залишився;
- Webpack, у процесі створення збірки, може мініфіціювати файли, що значно збільшує швидкість їх передачі, аналізу та виконання у браузері користувача.

3.5 Програмна платформа серверної частин

На боці користувача, традиційно, роль середовища виконання для веб-застосунку покладена на браузер (веб-переглядач, веб-навігатор) користувача, що запущено в операційній системі. Проте, для серверної частини вибір середовища виконання дуже великий. Найбільш популярний варіант – зв’язка логіки серверної частини, що написана на серверні мові програмування (а також інтерпретатора/компілятора цієї мови) та спеціалізованого серверного програмного забезпечення.

Для написання логіки роботи серверної частини використовуються такі мови як PHP, C#, Java, тощо.

На момент написання даного розділу існує три найбільш розвинуті варіанти спеціалізованого програмного забезпечення для реалізації веб серверів:

– IIS (Internet Information Services) – власницьке програмне забезпечення від компанії Microsoft;
– Apache HTTP Server – вільне, кросплатформне програмне забезпечення;
– Nginx – вільне програмне забезпечення, що розроблено під Unix-подібні операційні системи.

Проте, у даному проекті було вирішено використовувати інший варіант. Апаратна платформа NodeJS, що дозволяє писати серверну логіку мовою JavaScript, має також вбудовані можливості для реалізації веб-серверу. Таким чином, дві різні частини веб-застосунку реалізуються одним рішення, що дозволяє значно спростити та прискорити розробку. Також, завдяки можливості виконання коду мовою JavaScript на сервері, це дозволить скористатися технологією серверного рендерингу, що надає фреймворк VueJs.

За офіційною документацією, NodeJs – кросплатформна апаратна платформа, написана мовою C++, дозволяє запускати програмний код, написаний мовою JavaScript, поза середовищем браузера[9]. Вона має вбудований інтерпретатор та середовище виконання для мови JavaScript, а також засоби введення/виведення (та складної роботи із файлами) які із точки зору розробника виглядають як бібліотека, що вбудована у середовище NodeJs.

Для виконання JavaScript коду NodeJs використовує вільний рушій V8, що був розробленій компанією Google для використання у їх власному браузері.

NodeJs підтримує як асинхронність, так и багатопотоковість виконання програмного коду, що дозволяє гнучко оптимізувати складні розрахунки.

NodeJs побудовано за принципами парадигми подійно-орієнтованого програмування, що дозволяє значно підвищити швидкодію при високих навантаженнях на веб-сервер. За цією парадигмою, будь-яка дія, що виконується програмою, є відповіддю на певну подію. Прикладами подій можуть слугувати

дії користувача, події операційної системи, кінець довгої операції обчислення, отримання даних із віддаленого сервера, тощо.

3.5.1 Модульна структура

Як і сама мова JavaScript , NodeJs має модульну природу. Вона активно використовує модульну систему під назвою CommonJs, яка з’явилася до появи, у стандарті ECMAScript, модульної системи, і є невід’ємною частиною архітектури платформи NodeJs. Це несумісна зі стандартом система, тому використання її у клієнтській частині сервісу (браузері) небажане. Ця проблема може бути вирішена за допомогою спеціальних бібліотек, які можуть підміняти одну систему імпорту іншою.

«Модуль» є структурним поняттям. Разом із цим, замість терміну «модуль», інколи застосовуються терміни «бібліотека» чи «фреймворк», які більше акцентують увагу на логічному змісті модулю.

Невід’ємною частиною модульної природи NodeJs є своя власна вбудована система управління пакетами – npm. Пакети – це окремі компоненти програмного забезпечення. Кожен пакет із сховища npm являє собою один або кілька модулів системи CommonJs. Це дозволяє оптимізувати роботу, за рахунок підключення лише того функціоналу, який потрібен розробнику для виконання конкретних цілей. У поєднанні із вільним кодом платформи це забезпечує майже необмежений функціонал для розширення її можливостей, за рахунок можливості написання власних модулів будь-якими розробниками, та розповсюдження їх через npm. Тоді інші розробники, що будуть бажати скористатися вже готовими модулями, мають лише завантажити собі відповідні модулі (npm надає для цього зручний консольний інтерфейс) та імпортувати, за допомогою CommonJs, у потрібному місці коду потрібний модуль. Розглянемо кілька пакетів, що будуть використані для розробки майбутнього веб-

застосунку.

Варто зазначити, що npm має можливість встановлення модулів загально або локально. Перший варіант означає, що модуль буде встановлено як програму в операційну систему, і він буде доступній із командного рядка у будь-якому місці системи. Локальне встановлення навпаки, збереже модуль в окремому каталозі, що буде розміщений у каталозі де було запущено встановлення модулю. Доступ до нього буде можливий лише із даного каталогу. Локальне встановлення дає можливість тримати робочу операційну систему у чистоті, та уникати конфліктів, у разі якщо різні проекти потребують встановлення одного пакету, але різних версій.

Для великих проектів, npm дає можливість створювати конфігурацію у вигляді JSON файлу, де описуються назви та версії необхідних пакетів. Ці конфігурації зручно розповсюджувати між розробниками через системи контролю версій. За наявності конфігурації та встановленої системи керування пакетами, розробник має змогу, за допомогою однієї команди, встановити локально усі відповідні модулі потрібних версій, що згадуються у конфігурації.

Далі будуть розглянуті приклади модулів, що будуть активно використовуватися під час розробки.

3.5.1.1 Fs (розшифровується як file system) – модуль (бібліотека) введення/виведення, що надає розробнику можливість працювати із файловою системою. Він поставляється разом із платформою NodeJs, тому не потребує окремого встановлення через npm. Тут представлені усі необхідні функції для роботи із файлами (створення, редагування, читання, перевірка існування, тощо). За замовчуванням, майже усі функції виконуються асинхронно. Це означає, що викликаючи функцію, розробник одразу описує функцію зворотнього виклику, яка буде буде запущена бібліотекою по закінченню основної функції, та отримає результати цієї функції як параметри. Наприклад, при виклику функції читання файлу, розробник описує функцію зворотнього виклику, яка приймає вміст

файлу у якості параметру і буде запущена одразу після читання цього вмісту. Так як усі описані дії виконуються асинхронно, основна програма може виконувати інші дії у цей час, не блокуючи весь потік для читання файлу. Це класичний підхід подійно-орієнтованого програмування. Також, модуль fs надає синхронні аналоги функцій, для випадків, коли зупинка головного потоку виконання програми є бажаною поведінкою. Вони іменуються як асинхронні, але мають у кінці імені слово «Sync».

3.5.1.2 Express – модуль, який можна назвати повноцінним фреймворком. Його реалізовано на основі більш низькорівневого модулю http, що надає розробнику функціонал для роботи із однойменним протоколом[10]. Модуль http може працювати як у ролі клієнту, так і у ролі серверу. Express надає великий набір функцій для реалізації веб-серверу, від зовсім простого до високо навантаженого. Ось деякі з можливостей, абстракції для яких (поверх модулю http) надає модуль Express:

- обробка http запитів та відповідей;
- робота із формами;
- валідація даних, що отримані від клієнту;
- робота із cookies;
- робота із локальним сховищем даних браузеру;
- робота із асинхронними запитами JavaScript;
- планування виконання задач на певний час, чи з певним інтервалом.

Головна особливість модулю Express – підтримка проміжних (middleware) функцій. Це означає, що у кожному циклі «запит-відповідь» об’єкти запиту та відповіді проходять по черзі через список функцій визначених розробником. Визначені функції викликаються модулем Express у порядку, що визначено розробником. Кожній функція отримує, у якості параметрів, об’єкти запиту та

відповіді, і може їх змінювати. Наступна функція отримує об’єкти у тому вигляді, у якому вони були на момент завершення виконання попередньої функції. Кожна так функція, наприкінці свого виконання повинна передати керування назад до модулю, аби той викликав наступну функцію, або вона може вирішити завершити цикл обробки «запит-відповідь» та ініціювати відправлення відповіді клієнту. Таки чином реалізується конвеєрна обробка об’єктів запиту та відповіді. Це архітектурне рішення зручно поєднується із модульною природою NodeJs та npm, дозволяючи розробникам писати власні проміжні функції та розповсюджувати їх у вигляді пакету через npm.

3.5.1.3 TypeScript – вже описана вище технологія. Компілятор та супутні можливості TypeScript можна встановити, локально або глобально, як пакет npm. Також, відповідно локально або глобально, встановлюються файли декларацій для стандартних функцій мови JavaScript.

3.5.1.4 Обраний раніше веб-фреймворк VueJs, а також усі допоміжні бібліотеки, такі як VueRouter, Vuex та VueServerRenderer, розповсюджуються як окремі пакети npm. Кожен із них поставляється зі своїми власними файлами декларацій TypeScript.

3.5.1.5 Обрана для даного проекту програма для збірки модулів Webpack, може бути встановлена як пакет із сховища npm. Також можна окремо встановити пакети WebpackDevMiddleware та WebpackHotMiddleware, що надають проміжні функції для веб-серверу express, та використовують можливість Webpack-у слідкування за файлами. Їх комбінація значно прискорює розробку, дозволяючи одразу бачити,як на сервері так і на клієнтському боці, зміні, що були внесені у початкові фали.

3.6 Балансування навантажень на сервер

При рості кількості запитів до веб-застосунку на одиницю часу, постає питання збільшення пропускних здібностей серверу, який приймає усі запити та має надіслати кожному персональну відповідь.

Ця проблема може бути вирішена за рахунок покращення технічних характеристик обладнання, на якому працює веб-сервер. Наприклад обладнання його додатковим обсягом оперативної пам’яті, або заміна процесорів на більш потужні. Такі дії можуть допомогти виправити ситуацію, проте мають ряд серйозних недоліків. По-перше, серверне обладнання, як і уся фізична техніка, має свої обмеження, і не можливе нескінченне збільшення обчислювальних потужностей. По-друге, постає проблема із відмовостійкістю. При виході з ладу єдиної машини – весь величезний сервіс перестає працювати.

Враховуючи вищезазначене, для збільшення пропускної здібності прийнято використовувати масштабування не тільки вгору, а й вбік. Це означає створення декількох серверів або екземплярів серверного програмного забезпечення із однаковим функціоналом (це називають «кластер»). Це дозволяє ефективно використовувати паралельні обчислення, що значно збільшує пропускну здатність серверу, як цілої сутності, та підвищує відмовостійкість, бо при виході з ладу одного члена кластеру, інші візьмуть на себе його клієнтів.

Для балансування навантаження у даному проекті було вирішено використовувати веб-сервер nginx, який було розглянуто у попередньому підрозділі. Він підтримує функцію балансування, постійно слідкуючи за станом кластеру серверного обладнання та програмного забезпечення. Всі клієнтські запити надходять до nginx, а далі він перенаправляє їх до найменш зайнятих членів кластеру.

Балансування навантаження присутнє лише на етапі проектування, та не буде реалізовано у зв’язку із обмеженим часом, що відведено на розробку

проекту.

3.7 Сховище даних

Так як цілісність даних є однією із головних вимог до веб-застосунку, що розробляється у рамках даного проекту, та дані можуть бути чітко розділені по сутностям та зв’язкам між ними (категорії товарів, товари), то було прийнято рішення зберігати дані у зовнішньому сховищі у форматі реляційної бази даних. Бази даних, що базуються на реляційній моделі даних можуть надійно забезпечити цілісність даних та демонструвати гарну продуктивність, при роботі із чітко структурованими, розділеними на сутності, даними.

Додатковою перевагою використання реляційних баз даних є те, що всі вони гарантовано підтримують роботу через запити мовою SQL (Structured query language). SQL – декларативна мова програмування, розроблена спеціально для взаємодії (створення, видалення, зміни, тощо) із даними у базах даних, що базуються на реляційній моделі даних[11]. Фактично, SQL складається із чотирьох частин, що називаються окремими мовами:

- DDL (Data definition language) включає оператори SQL, що відповідають за взаємодію із об’єктами бази даних;
- DML (Data manipulation language) включає оператори SQL, що відповідають за взаємодію із даними в об’єктах бази даних;
- TCL (Transaction control language) включає оператори SQL, що відповідають за проведення транзакцій у базі даних;
- DCL (Data control language) включає оператори SQL, що відповідають за взаємодію із корстувачами та права доступу користувачів до бази даних.

3.7.1 Система управління базами даних

Усі існуючі системи управління реляційними базами даних підтримують взаємодію за допомогою мови SQL, проте відрізняються внутрішніми алгоритмами роботи, швидкістю виконання тих чи інших операцій, алгоритмами оптимізації та нормалізації баз даних, ризнии наборами типів даних, що підтримуються. Найтиповішими представниками реляційних систем управління базами даних (РСУБД) є:

- MySql. Найпопулярніша на ринку РСУБД, яка є дуже швидкою, проте не повністю реалізує можливості SQL і вважається «полегшеною» системою управління для невеликих проектів. База даних MySql повністю зберігається в одному файлі, що її полегшує перенос з однієї машини на іншу та створення резервних копій. Із недоліків варто зазначити, що MySql має проблеми із забезпеченням цілісності даних, внаслідок полегшеного механізму транзакцій.
- Microsoft SQL Server. РСУБД від компанії Microsoft, що має повну реалізацію SQL, а також власне розширення цієї мови, під назвою Transact-SQL. Дана РСУБД може повністю забезпечити цілісність даних, проте працює повільніше ніж MySql, та потребу великого обсягу пам’яті, порівняно з іншими РСУБД, для зберігання такого самого обсягу даних;
- PostgreSQL. Найновіша серед РСУБД, що розглядається у даному розділі. Забезпечує повну цілісність даних, дещо поступаючись при цьому MySql, проте значно випереджаючи рішення від Microsoft. Підтримує хберігання даних у стисненому вигляді, що зменшує вимоги до обсягу пам’яті для бази даних.

Для даного проекту було обрано реляційну систему управління базами даних PostgreSQL, так як вона забезпечує оптимальний баланс між надійністю, швидкодією та обсягом пам’яті, що потребується.

PostgreSQL є кросплатформною і має драйвери для роботи із більшістю сучасних операційних систем та програмних платформ.

Висновки до розділу 3

У даному розділі було обрано та описано усі програмні інструменти, що будуть використовуватися під час розробки веб-застосунку, який є основною метою даного проекту.

Усі обрані технології є вільним та відкритим програмним забезпеченням, що означає можливість безкоштовного їх використання, навіть для розробки закритих комерційних проектів. Також це дає можливість, за необхідності, вносити зміни у програмний код самих інструментів для налаштування їх під розробку конкретного проекту.

Обраний стек технологій дозволить швидко та якісно розробити майбутній веб-застосунок.

4 РОЗРОБКА АРХІТЕКТУРИ ВЕБ-ЗАСТОСУНКУ

Перший крок у розробці будь-якого програмного забезпечення – розробка його архітектури. У термінології розробки програмного забезпечення, термін «архітектура» має дуже розпливчате значення. Загалом, термін «архітектура» є наближеним до терміну «проектування», якщо казати про структурне проектування, а не проектування коду.

Якщо його узагальнити, то під цим терміном варто розуміти наступні етапи:

- розбиття майбутнього програмного забезпечення на мінімальні структурні елементи, що необхідні для його реалізації;
- послідовне об’єднання визначених структурних елементів у логічні групи, доки не буде отримано єдину цілісну систему систему;
- опис структур даних, що будуть використовуватися для взаємодії між структурними елементами, або їх групами;
- опис зв’язків між структурними елементами, або їх групами (напрями потоків даних між елементами та їх зміст);
- опис правил та обмежень, яких слід дотримуватися при написані коду та при розгортанні готового застосунку.

Додатково, розширив поняття архітектури до початку розробки, можна додати останнім пунктом вибір для кожного структурного елемента чи їх логічної групи відповідної технології програмного забезпечення, або прийняття рішення про необхідність власної реалізації такої технології у рамках програмного забезпечення, що розробляється.

Таким чином, визначимо, що архітектура програмного забезпечення – набір, фундаментальних для проекту, рішень та правил, що пов’язані із фізичною та програмною структурами проекту. Від цих рішень залежить

максимальна швидкодія проекту, його надійність та здатність до розширення.

Якісно спроектована архітектура – архітектура, що дозволяє повністю забезпечити виконання вимог до програмного забезпечення щодо користувацьких можливостей, швидкодії, надійності програмного забезпечення, та дозволяє легко розвивати програмне забезпечення, додаючи нові можливості його використання.

Існують, так звані, архітектурні стилі (архітектурні шаблони). Це узагальнені принципи побудови архітектури програмного забезпечення, що гарно показали себе у багатьох різних рішеннях, та на які можна спиратися при проектуванні власної архітектури. Розберемо найпопулярніші з них, на момент написання даного розділу, спираючись на статті [12, 13]:

– сервіс-орієнтована архітектура. Дуже загальний підхід, що полягає у розділенні (фізичному та/або логічному) логічно різних структурних компонентів у окремі сервіси, які інкапсулюють певну логіку. Такий підхід дозволяє покращити загальну завадостійкість програмної системи, за рахунок того, що при відмові одного із сервісів буде втрачено лише його функціонал, а інші сервіси продовжуватимуть взаємодію між собою. Також це дозволяє легше змінювати структуру системи, за рахунок слабкої зв’язності та абстракцій, якими слугують інтерфейси сервісів;

– мікросервісна архітектура. Під даним терміном розуміють ідеї сервіс-орієнтованої архітектури, що розвинені в абсолют. Архітектура будується за рахунок максимальної кількості мінімальних за логічним функціоналом сервісів, що взаємодіють між собою;

– архітектура, що спирається на події. Основна ідея цього шаблону полягає у розділенні програмної системи на структурні компоненти, які не взаємодіють між собою напряму, але можуть самі ініціювати певні події, та реагувати на події від інших структурних компонентів. Вся система будується на використанні шаблону проектування «публікація-підписка». Дана архітектурний стиль, як і сервіс-орієнтована архітектура прагне до

послаблення зв’язків між частинами системи, задля більшої відмовостійкості, більших можливостей до змін та розширення;

– клієнт-серверна архітектура. Для деяких програмних систем, де актуальна інформація може часто змінюватися, зручно розділяти структуру на постачальників деяких послуг та споживачів цих самих послуг. Ці елементі називаються, відповідно, «сервер» та «клієнт». Клієнти мають змогу запитувати у серверів певні дані, а сервери постійно прослуховують запити клієнтів та відправляють у відповідні необхідні дані. Це дозоляє оптимально розділяти обчислювальні потужності між відповідними сторонами, та балансувати навантаження на сервери, шляхом їх дублювання та перенаправлення клієнтів на різні сервери. Головним недоліком даної архітектури є одностороння залежність програмної системи від серверів, що не дозволяє казати про високий показник відмовостійкості. При виході серверу із ладу, послуга, що він постачав, втрачається. Для запобігання цьому використовують дублювання серверів, щоб при виході із ладу одного, клієнти могли звертатися до іншого. Даний архітектурний стиль лежить в основі роботи основних протоколів мережі Інтернет (HTTP, FTP, тощо);

– однорангова архітектура. Виступає як певна протилежність архітектурі «клієнт-сервер». Проводячи аналогію, кожен структурний елемент (учасник) може виступати у ролі серверу та клієнту одночасно. Таким чином, усі елементи програмної системи є рівноправними. Цей стиль забезпечую максимальну відмовостійкість (сервіс може працювати, допоки є хоча б один учасник). Прикладами мереж, що побудовані за принципами однорангової архітектури можна назвати BitTorrent та Gnutella.

Кожен із архітектурних шаблонів не є однозначним правилом. Для кожного проекту їх можна и потрібно видозмінювати та комбінувати, щоб спроектувати архітектуру, що якнайкраще підходить до вимог кожного окремого випадку.

Для опису готових архітектурних рішень часто використовують мови графічного опису архітектури (ADLs). Такі спеціалізовані графічні мови мають великі можливості, проте часто виявляються занадто перевантаженими для зручного розуміння. У цих випадках зручно використовувати універсальну мову графічного опису UML (Unified Modeling Language), що розроблена спеціально для сфері розробки програмного забезпечення. У даному проекті буде частина графічної документації, що зображує архітектуру застосунку, буде розроблена за допомогою засобів мови UML.

4.1 Структура веб-застосунку

Для веб-застосунку, що розроблюється у рамках даного проекту було вирішено комбінувати кілька архітектурних стилів, що гармонічно доповнюють один одного.

Сама концепція веб-застосунку означає, що сервіс буде базуватися на клієнт-серверній архітектурі, для роботи у мережі Інтернет.

Серверна частина буде реалізована із використання сервіс-орієнтованої архітектури. Основним сервісом буде серверне програмне забезпечення, що представлене апаратною платформою NodeJs. Цей сервіс відповідатиме за доставку програмного коду клієнтської частини до браузерів, у відповідь на запити клієнта на інтернет-адресу, яку буде мати веб-застосунок.

У майбутньому планується об’єднання запущених екземплярів NodeJs у кластери, та керування ними через nginx. Це є теоретичною частиною проектування веб-застосунку и буде відображено у графічній документації до проекту. Проте розробка цього функціоналу виходить за межі даної роботи, у зв’язку із обмеженням у часі.

Як окремі сервіси будуть працювати сховища даних. У даному проекті реалізується два сховища. Перше буде містити інформацію про каталог

категорій та товари у ньому. Інформація із нього буде доступна серверам та клієнтам за допомогою API, що виходить у мережу Інтернет. Друге сховище буде використовуватися виключно для зберігання зображень. Ним буде користуватися виключно клієнтська сторона, завантажуючи зображення за URL адресою. На стороні серверу інформація про зображення буде представлена тільки URL адресами, що отримані з API першого сховища.

Для збору інформації має використовуватися окреме фізичне серверне обладнання із апаратною платформою NodeJs. На ньому мають автоматизовано запускатися скрипти для збору інформації, та завантажувати оновлені дані у сховища даних.

Клієнтська частина буде реалізована у вигляді єдиного односторінкового веб-застосунку із використанням веб-фреймворку VueJs.

Для більш детального уявлення про логічну та фізичну структури проекту, було розроблено структурну схему веб-застосунку та діаграму розгортання мовою UML. Вони містяться у ілюстративних матеріалах проекту.

4.2 Схема бази даних

У даному підрозділі описано сутності, що містяться у сховищі даних. Повна схема бази даних міститься у ілюстративних матеріалах проекту.

У базі даних зберігається інформація про категорії (як загальна інформація, так і ієрархія дерева категорій) та про товари. База даних розрахована на збереження інформації одразу на двох мовах (державні мови Канади – англійська та французька).

4.2.1 Перелічувані типи даних

Перелічуваний тип даних являє собою визначений перелік (набір) строкових констант (ідентифікаторів). Значенням полю, що має даний тип, є один ідентифікатор із переліку. Для мов програмування, що реалізують даний тип даних, перелічувані типи даних це набір цілочисельних констант, а строкові ідентифікатори, що закріплені за ними, є абстракцією для зручності розробників, що користуються цим типом даних. У схемі бази даних даного проекту існує два перелічувані типи даних :

- Language. Тип перелічення, що зберігає інформацію про мову, якою зберігається інформація у тому чи іншому кортежі. Має два значення. Ідентифікатор «en», від англійського «English», що і означає «англійська». Та «fr», від англійського «French», що означає «французька». Винесення мови в окремий перелічуваний тип даних дозволяє у майбутньому легко розширювати список мов, що підтримуються;
- ImageType. Хоча інформація про зображення і зберігається у базі лише як URL посилання, проте інша інформацію, про відношення зображення до товарів, повинна міститися у базі даних Перелічення ImageType показує призначення зображення, та має три значення. Ідентифікатор «main» означає, що у даному кортежі міститься інформація про головне зображення товару, що виводиться на картках товарів у каталозі та є першим у галереї зображень на власній сторінці товару. Ідентифікатором «gallery» позначені зображення, що присутні у галереї на власній сторінці товару, за винятком головного (першого). Останній з ідентифікаторів, «listing», позначає зменшені копії усіх зображень із галереї на власній сторінці товару, що користувач бачить при попередньому перегляді.

4.2.2 Відношення

За описом Мірошниченко Г. А., «відношення» є одним із основних

термінів реляційної моделі даних, за якою побудовані реляційні бази даних[14]. Відношення це скупність заголовку, що має набір атрибутів, та тіла, що являє собою набір кортежів. Кожна сутність у базі даних являє собою одне відношення. Для спрощення, відношення зазвичай представляють у вигляді таблиць, а кортежі у вигляді рядків таблиці. Атрибути, заголовки яких містяться у заголовку відношення, а значення – у кортежах, представляють у вигляді стовпців таблиці. Отже, можна казати, що далі у пункті будуть розглянуті сутності, що належать схемі бази даних. Спочатку буде розглянуто відношення, що пов’язані із категоріями, а далі – відношення, що пов’язані із товарами.

4.2.2.1 Перше відношення, яке варто оглянути, у контексті категорій, представляє джерело даних. Воно має назву «marketpalces». Має атрибути унікального ідентифікатору (первинний ключ, що автоінкрементується) та назви (тип даних текст).За самі категорії відповідають два відношення.

Перше із них представляє інформацію, що не залежить від мови. Має назву «categories». Має два атрибути ідентифікаторів. Ідентифікатора джерела даних, із якого взята категорія, що забезпечує зв’язок із відношенням marketpalces, та власний числовий ідентифікатор. Також присутній атрибут, що містить числового ідентифікатор батьківської категорії («parentId»). За допомогою даного атрибуту зберігається інформація про положення категорії у ієрархічному дереві.

Друге відношення, що містить інформацію про категорії, має назву «categories_lc» та відповідає за частину інформації, залежну від мови. Має зв’язок із відношеннями categories через атрибути ідентифікатору джерела даних та ідентифікатору категорії. Також має ідентифікатор мови (атрибут має перелічуваний тип language). У цьому відношенні присутні два текстові атрибути, що містять ім’я категорії ті її опис («name» та «description», відповідно). На кожен кортеж у відношенні категорій приходиться по кілька кортежів із відношення локалізації категорій, де «кілька» дорівнює кількості

ідентифікаторів перелічення мов.

4.2.2.2 Зв’язуючою ланкою між категоріями та товарами виступає відношення, що має назву «product_category». Воно реалізує зв’язок «багато до багатьох». Даний зв’язок виникає у наслідок присутності одного й того ж товару у різних джерелах даних. Містить три зовнішні зв’язки через наступні атрибути для зберігання ідентифікаторів:

– числовий ідентифікатор джерела даних;
– числовий ідентифікатор категорії;
– числовий ідентифікатор товару.

Найбільше відношення, що пов’язане із товарами, має назву «products». Зв’язок із іншими відношеннями забезпечується атрибутами для ідентифікатору джерела даних та власного текстового ідентифікатору товару. Окрім них, дане відношення містить наступні атрибути:

– categoryId – числовий ідентифікатор категорії, до якої належить даний товар;
– url – URL посилання на сторінку товару на веб-сайті компанії, що виготовляє даний товар;
– brand – назву компанії, що виготовляє даний товар;
– brandLogoUrl – URL посилання на зображення логотипу компанії, що виготовляє даний товар;
– brandStoreLink – URL посилання на веб-сайт компанії, що виготовляє даний товар;
– customerRating_count – кількість користувачів, що поставили оцінку даному товару;
– customerRating_score – середня (медіанна) оцінка даного товару

користувачами;

- isDiscount – булеве значення, що показує чи є акційна пропозиція на даний товар;
- priceRegular – ціна на товар, без урахування можливих акційних пропозицій;
- priceDiscount – ціна на товар при наявності акційної пропозиції;
- priceCurrent – актуальна ціна на товар, що. Дорівнює акційній ціні, за наявності акційної пропозиції, або звичайній ціні , за відсутності акційних пропозицій;
- saleEndDate – містить дату закінчення поточної акційної пропозиції, за її наявності.

Окрім великого відношення products, є декілька відношень, у яких зберігається додаткова інформація.

Відношення «product_lc», представляє частину інформації про товар, що залежить від мови. За структурою це відношення подібне до аналогічного відношення для категорій. Зовнішні зв’язки виражаються атрибутами для ідентифікатору джерела даних, ідентифікатору товару та ідентифікатору мови. Дані містяться у текстових атрибутах «name», «shortDesription» та «longDescription». Перше містить назву товару. Друге – короткий опис, що використовується при попередньому перегляді. У третьому атрибуті зберігається текст повного опису товару, що виводиться тільки на персональній сторінці товару. Відповідно, для кожного товару у даному відношенні буде стільки кортежів, скільки ідентифікаторів присутні у переліченні мов.

Відношення «product_images» зберігає інформацію про усі зображення, що відносяться до товарів, та знаходяться у окремому сховищі даних. Зв’язок із іншими відношеннями забезпечується атрибутами ідентифікатору джерела даних, ідентифікатору товару, ідентифікатору типу зображення (атрибут має перелічуваний тип ImageType) та числового атрибуту «orderNum», що зберігає

інформацію про порядок слідкування зображень у галереї на власній сторінці товару. У даному відношенні ще містяться текстові атрибути «url» та «thumbnailUrl», які зберігають URL посилання, відповідно, на повну та зжату (зменшену для попереднього перегляду) версії зображення. Також присутні числові атрибути «width» та «height», що зберігають, відповідно, ширину та висоту зображення у пікселях.

Відношення «product_options» зберігає інформацію про один і той самий товар,що може існувати у різних конфігураціях. Зовнішні зв’язки виражаються атрибутами ідентифікатору джерела даних, ідентифікатору товару та власного текстового ідентифікатору поточної конфігурації. Конфігурація зберігається у кортежах як словник пар «ключ-значення». Кожен ключ та кожне значення мають також власні текстові ідентифікатори для розпізнавання одних і тих самих пар різними мовами. Крім атрибутів первинного ключа дане відношення має наступні атрибути:

- атрибути url, isDiscount, priceRegular, priceDiscount та priceCurrent – відповідають атрибутам із аналогічними назвами, що містяться у відношенні products;
- lang – атрибут, що має перелічуваний тип даних Language. Відповідає за мову кортежу;
- optionKeyId – текстовий ідентифікатор назви ключа елементу конфігурації;
- optionValueId – текстовий ідентифікатор поточного значення елементу конфігурації;
- optionKey – назва ключа елементу конфігурації;
- optionValue – поточне значення елементу конфігурації.

Останнє із розглянутих відношень має назву «raw_specification_values». Воно пов’язане із іншими відношеннями за допомогою атрибутів ідентифікатору джерела даних, ідентифікатору товару та ідентифікатору мови. Воно зберігає інформацію про характеристики (специфікацію) товару у вигляді словнику пар

«ключ-значення». Кожен ключ та кожне значення мають також власні текстові ідентифікатори для розпізнавання одних і тих самих пар різними мовами. Відношення включає наступні атрибути:

- mpId – ідентифікатор джерела даних;
- productId – ідентифікатор товару;
- orderNum – числовий атрибут, що зберігає інформацію про порядок слідкування характеристик у специфікації;
- extAttributeId – текстовий ідентифікатор назви ключа елементу специфікації.
- extValueId – текстовий ідентифікатор поточного значення елементу специфікації;
- displayName – назва ключа елементу специфікації;
- value – поточне значення елементу специфікації.

4.3 Модель даних у коді веб-застосунку

Усі дані, що описані у попередньому розділі, використовуються клієнтською частиною веб-застосунку, а отже повинні зберігатися у ній, під час роботи. Для зручності використання даних у програмному коді, модель даних у ньому дещо відрізняється від тієї, за якою побудована база даних.

Модель даних у коді веб-застосунку, як і схема бази даних, також має принципове розділення на сутності «категорій» та «товарів».

Варто відзначити, що було прийнято рішення не включати до структур даних у коді ідентифікатор мови, оскільки зміна мови є дуже рідко використовуваною можливістю (кожен клієнт, зазвичай, один раз вибирає зручну для нього мову та не змінює її). Тому одночасно зберігаються лише

інформація однією мовою, що на даний момент обрана клієнтом у налаштуваннях веб-застосунку. При зміні мови, сторінка веб-застосунку оновлюється «на місці» так, наче користувач почав нову сесію роботи із веб-застосунком.

Треба зазначити різницю, що у структурах даних у коді є лише один власний числовий ідентифікатор, що представляє кожен об’єкт. Цей ідентифікатор створюється на етапі перетворення даних із структури, у якій вони зберігаються, у структури, що будуть передані до веб-застосунку через API.

У додатку А наведена діаграма класів мовою UML для моделі даних коду веб-застосунку.

4.3.1 Категорії представлені єдиним інтерфейсом, що моє назву «Category». Він має наступні поля:

- id – власний числовий ідентифікатор категорії;
- name – назва категорії;
- parentId – власний числовий ідентифікатор батьківської категорії до даної (для категорій, що є кореневими у ієрархічному дереві, це поле містить спеціальне значення-маркер «null»);
- parents – масив, що містить послідовні посилання на об’єкти батьківських категорій. Будується рекурсивно, перше посилання буде на категорію, що має власний ідентифікатор із поля parentId, а останнє буде посилатися на одну із кореневих категорій;
- children – масив, що містить послідовні посилання на об’єкти дочірніх категорій;

Останні два поля (parents та children) не є обов’язковими, та формуються тільки за необхідності (окремим запитом до API), щоб уникати великої кількості

кільцевих посилань.

4.3.2 Товари представлені декількома інтерфейсами. Частина с них схожа на відповідно структури у базі даних, а деякі створені тільки у коді веб-застосунку, щоб полегшити роботу із даними.

Для товарів, окрім допоміжних інтерфейсів, спроектовано два основних. Один із яких наслідує (має всі ті ж самі поля, та ще додаткові) інший. Це зроблено внаслідок того, що для попереднього перегляду товарів потрібна менша кількість інформації, ніж та, що присутня на власній сторінці товару.

Спочатку буде розглянуто допоміжні інтерфейси, що необхідні для проектування головних.

Перший розглянутий інтерфейс має назву «ProductImage». Об’єкти цього типу зберігають інформацію про зображення, що відносяться до товарів. Інтерфейс містить поле «id», що зберігаю унікальний числовий ідентифікатор даного зображення. Він формується при отриманні даних із бази даних, із значень первинного ключа відповідного кортежу відношення product_images. Відсутній також аналог атрибуту orderNum, а порядок слідкування зображень формується на основі порядку слідкування об’єктів у масиві, що отриманий із віддаленого сховища. У процесі переформавання структури даних, цей масив заповнюється у порядку, що відповідає значенню атрибуту orderNum. Інші поля інтерфейсу повністю відповідають значенням однойменних атрибутів із бази даних: url, thumbnailUrl,width та height.

Два інтерфейси із назвами «ProductLocalized« та «ProductLocalizedShort» містять ту інформацію про товари, що залежить від мови користувача. Дана інформація винесена у окремо структуру із метою зменшення кількості даних, що потребують оновлення, при зміні користувачем мови інтерфейсу веб-застосунку. Інтерфейс ProductLocalizedShort містить текстових поля «name» та «shortDescription», що зберігають, відповідно, ім’я товару та короткий опис для

попереднього перегляду. Інтерфейс ProductLocalized наслідує ProductLocalizedShort та містить власне текстове поле «longDescription», де зберігається повний (детальний) опис товару для відображення на його власній сторінці.

Менший, із головних інтерфейсів товару, має назву «ProductShort» та містить наступні поля:

- id – власний числовий ідентифікатор товару;
- mainCategoryId – ідентифікатор категорії, до якої належить товар;
- minPrice – найменша ціна на товар серед усіх продавців, що його пропонують;
- maxPrice – найбільша ціна на товар серед усіх продавців, що його пропонують;
- isDiscount – булеве значення, що показує чи є акційна пропозиція на даний товар;
- localized – поле, що містить об’єкт типу ProductLocalizedShort.
- rating – середня (медіанна) оцінка даного товару користувачами;
- mainImage – поле, що містить об’єкт типу ProductImage, який зберігає інформацію про головне зображення товару, яке міститься на сторінці попереднього перегляду (у каталозі);

Головний інтерфейс, для зберігання інформації про товари, має назву «Product». Він наслідує інтерфейс ProductShort, та містить наступні власні поля:

- model – назва конкретної моделі (конфігурації) товару, для товарів із однаковим іменем, що представлені у різних конфігураціях;
- vendor – назва компанії, що виготовляє даний товар;
- listImages – поле, що містить масив об’єктів типу ProductImage, у яких зберігаються зменшені копії зображень товару, що знаходяться на власній

сторінці товару;

– galleryImages – поле, що містить масив об’єктів типу ProductImage, у яких зберігаються версії повного розміру зображень товару, що знаходяться на власній сторінці товару;
– localized – поле, що містить об’єкт типу ProductLocalized.
– ratingCount – кількість користувачів, що поставили оцінку даному товару;

4.4 Модель даних Flux сховища

У попередньому розділі було прийнято рішення використовувати бібліотеку Vuex, що є частиною екосистеми VueJs для створення сховища на клієнтській стороні на час роботи із веб-застосунком.

Якщо роздивитися архітектуру Vuex більш детально, можна відмітити, що ця бібліотека розширює можливості стандартної архітектури Flux.

Для отримання даних у певному форматі Vuex надає можливість використовувати гетери, що оперують даними зі сховища. Вони є аналогом представлень із архітектури Flux, та мають вбудований механізм кешування. Це означає, що при повторному зверненні до гетера, якщо із часу минулу звернення стан сховища не змінився, значення не буде обчислюватися повторно, а розробнику буде повернено попереднє значення.

Важлива частина архітектури Vuex є розділення концепції «дій», із архітектури Flux, на дві концепції «мутації» та «дії» (далі у підрозділі, під «діями» буде матися на увазі саме «дії» у розумінні архітектури Vuex, якщо не зазначено інше).

Мутації створені для зміни даних у сховищі. Це атомарні операції, які повинні виконуватися строго синхронно для забезпечення цілісності та послідовного доступу до даних сховища.

Дії створені для розширення можливостей мутацій. Вони не можуть працювати напряму зі сховищем, а лише ініціювати мутації. Одна дія може ініціювати будь-яку кількість мутацій. Також на дії не розповсюджується обмеження мутацій на синхронність. Вони є асинхронними за замовчуванням, що дозволяє оптимізувати швидкість роботи веб-застосунку, зберігаючи при цьому усі переваги Flux архітектури.

Ще одна велика перевага бібліотеки Vuex, це можливість розподілення сховища на модулі. Завжди є кореневе сховище, що створюється при старті застосунку. Кореневе сховище може мати посилання (поля сховища) на інші об’єкти, які мають не тільки власний стан даних, а й власні гетери, дії та мутації. Таким чином, структура сховища застосунку може мати структуру фракталу, що дозволяє зручно зберігати структуровані дані.

Для даного веб-застосунку було розроблено чотири модулі, кожен із яких зберігає інформацію, що необхідна окремим частинам функціоналу веб-застосунку.

4.4.1 Перший модуль має назву «categoriesTree», та призначений для зберігання інформації, що необхідна для навігації. У ньому проходить процес кешування ієрархічного дерева категорій. Сховище даного модулю являє собою об’єкт словнику, що зберігає пари «ключ-значення». Ключем виступає власний числовий ідентифікатор категорії, а значенням є посилання на сам об’єкт у пам’яті.

Відмова від використання стандартної структури типу «масив» зумовлена необхідністю частої перевірки, чи є вже кешована та чи інша категорія. Операція пошуку по ключу у словнику (пошук у відсортованому масиві) значно швидша за пошук певного об’єкту у масиві (повний перебір масиву), що дає значний виграш у швидкості роботи зі сховищем.

Модуль має один гетер, що зберігає функцію, яка приймає один числовий

параметр. Гетер повертає булеве значення, що відповідає наявності категорії із ідентифікатором, що дорівнює переданому числу, у сховищі.

У модулі реалізовна єдина мутація, що приймає один об’єкт категорії, та додає його у сховище. числовий параметр, та додає у сховище категорію із ідентифікатором, що дорівнює отриманому параметру.

Єдина наявна у модулі дія є асинхронною обгорткою для мутації. Вона приймає числовий параметр і викликає гетер із цим параметром. При негативному результаті (відсутність категорії із ідентифікатором рівним параметру) вона посилає запит на API, передаючи ідентифікатор та поточну, обрану користувачем, мову веб-застосунку. Після цього дія ініціює мутацію, передаючи до неї отриманий із віддаленого серверного сховища об’єкт. Таким чином, категорія, що один раз потрапила до сховища, кешується у ньому до кінця сеансу роботи.

4.4.2 Модуль із назвою «wishlist» відповідає за зберігання інформації про список «бажаних» товарів користувача.

Сховище представляю собою звичайний масив об’єктів товарів, типу ProductShort.

Модуль має один гетер, що зберігає функцію, яка приймає один числовий параметр. Гетер повертає булеве значення, що відповідає наявності товару із ідентифікатором, що дорівнює переданому числу, у сховищі.

Модуль має три мутації:

- мутація, що приймає об’єкт товару та додає його до масиву у сховищі;
- мутація, що приймає числове значення, та видаляє із масиву товарів у сховищі об’єкт із відповідним індексом;
- мутація, що повністю очищає масив (видаляє список «бажаного»).

Три дії, що реалізовано у модулі відповідаю трьом мутаціям, проте

приймають не поодинокі значення, а масиви:

- дія, що приймає масив об’єктів товару та для кожного викликає гетер, перевіряючи чи є даний товар об’єкт у сховищі. При його відсутності, ініціює мутацію додавання товару, передаючи їй відповідний об’єкт;
- дія, що приймає масив числових значень та для кожного шукає індекс об’єкту у сховищі, що має такий самий ідентифікатор. При знаходженні, ініціює мутацію видалення товару, передаючи їй відповідний індекс;
- дія, що ініціює мутацію повного очищення масиву у сховищі.

4.4.3 Модуль із назвою «compare» схожий на попередній і відповідає за зберігання інформації про списки для порівняння товарів. Головна різниця полягає у тому, що списки для порівняння повинні групуватися по категоріям, тому сховище даного модулю являє собою масив об’єктів спеціально розробленого типу «CompareList». Цей тип включає два поля. Перше містить урізаний об’єкт категорії, що включає числовий ідентифікатор та ім’я категорії. Друге поле є масивом об’єктів товарів, що включені до списку порівняння у відповідній категорії.

Модуль має два гетери. Перший зберігає функцію, яка приймає один числовий параметр. Він повертає булеве значення, що відповідає наявності товару із ідентифікатором, у будь-якому зі списків у сховищі. Другий зберігає функцію, яка приймає один числовий параметр. Потім шукає об’єкт товару що має ідентифікатор рівний прийнятому параметру у будь-якому зі списків у сховищі. Повертає гетер ідентифікатор категорії, у списку якого є потрібний товар..

Модуль має чотири мутації:

- мутація, що приймає об’єкт категорії та створює новий список для порівняння за цією категорією;

– мутація, що приймає об’єкт товару, та додає його до списку порівняння за категорією, які є батьківською для даного товару;

– мутація, що приймає ідентифікатори товару та категорії і видаляє товар із відповідним ідентифікатором із списку категорії, що має переданий ідентифікатор.;

– мутація, що приймає ідентифікатор категорії та видаляє об’єкт категорії, що відповідає отриманому ідентифікатору, зі сховища.

Три дії, що реалізовано у модулі відповідаю групують мутації наступним чином:

– дія, що приймає масив об’єктів товарів. Для кожного із об’єктів виконується наступна послідовність кроків. Дія шукає його у списках, викликаючи відповідну функцію із гетеру. Якщо даного товару не було знайдено, перевіряється чи є список порівняння за категорією, до якої належить товар. Якщо відповідного списку не знайдено, то ініціюється дія із модулю categoriesTree для отримання інформації про ім’я категорії до якої належить отриманий товар. Далі ініціюється мутація для створення нового списку порівняння за категорією. Після цього ініціюється мутація для додавання товару у список, що був щойно створений, або вже існував;

– дія, що приймає масив числових значень. Для кожного зі значень виконується наступна послідовність кроків. Дія шукає товар із відповідним ідентифікаторам по всім спискам. Якщо відповідний товар знайдено, то ініціюється мутація для видалення об’єкту товару зі списку. Далі перевіряється чи залишилися ще об’єкти у даному списку. Якщо був видалений останній об’єкт, то ініціюється мутація для видалення усього об’єкту списку порівняння;

– дія, що приймає числовий параметр. Вона шукає у сховищі список порівняння, категорія якого має ідентифікатор, що дорівнює отриманому параметру. При знаходженні, ініціюється мутація видалення знайденого

об’єкту списку.

4.4.4 Vuex може взаємодіяти із рішенням для маршрутизації VueRouter, яке було розглянуто у попередньому розділі. За допомогою спеціального модулю VueRouterSync, що надають розробники екосистеми фреймворку VueJs, стан маршруту може бути вбудовано як окремий модуль сховища веб-застосунку. Дане архітектурне рішення дозволяє гетерам, діям та мутаціям отримувати інформацію про поточне місцезнаходження користувача у веб-застосунку.

4.5 Модуль збору даних

У розділі 2 було прийнято рішення про необхідність написання модулю для автоматизованого збору даних. Даний модуль повинен надавати каркас для написання однотипних скриптів під кожне джерело даних веб-застосунку.

Спрощену структуру модулю зображено на рисунку 4.1.

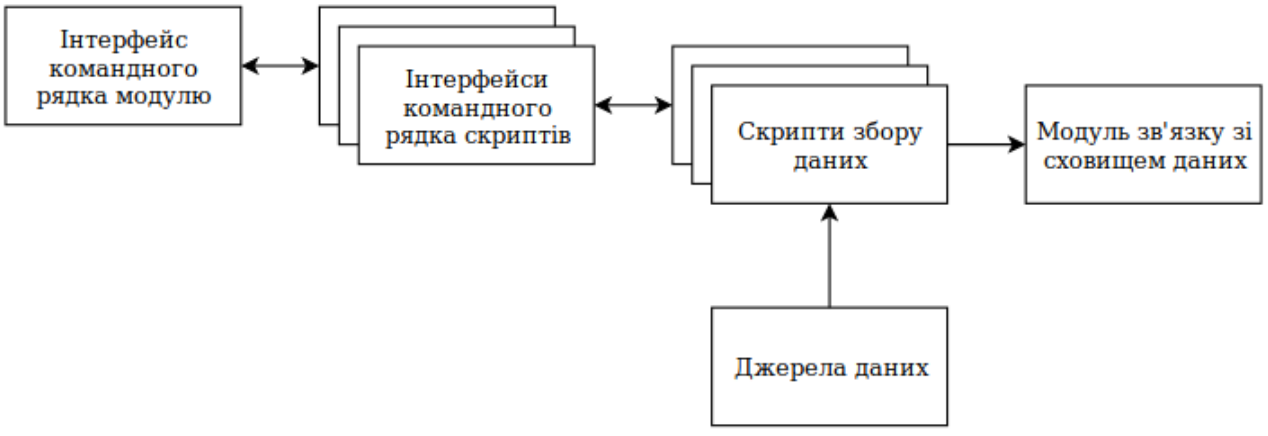

Рисунок 4.1 – Структурна схема модулю для збору даних

Для зовнішнього та внутрішнього управління було вирішено використовувати інтерфейси командного рядка, за наступними критеріями:

– даний модуль призначений, у першу чергу, для розробників та адміністраторів. Це виключає необхідність графічного оформлення, залишаючи лише вимогу зробити інтерфейс функціональним;

– простота у реалізації із точки зору розробника, порівняно із графічним інтерфейсом;

– робота із інтерфейсом командного рядка легко автоматизуються. Більшість операційних систем мають вбудовані механізми автоматизованого запуску скриптів для командного рядку за часом.

Для створення каркасу командного рядка було використано бібліотеку yargs із арсеналу npm. Вона дозволяє визначати власні команди для командного рядка, виклик яких ініціює запуск закріплених за цією командою скриптів у середовищі NodeJs.

Кожен скрипт має власний інтерфейс командного рядка, який містить одну основну команду, що запускає увесь процес збору даних. Також він може містити додаткові команди для збору окремих наборів даних (наприклад, дані тільки по товарах, або тільки однією заданою мовою).

Головна командна загального інтерфейсу одночасно викликає команди збору інформації для кожного джерела даних у різних потоках. Багатопотокова архітектура модулю дозволяє значно пришвидшити збір усієї необхідної інформації.

Варто детальніше розглянути сам механізм збору даних. Кожна команда окремого інтерфейсу командного рядка для джерела даних запускає створені розробником скрипти для збору даних. Як і набори скриптів, кожен окремий скрипт запускається у своєму потоці. Таким чином, усі скрипті скрипти виконуються одночасно, дозволяючи досягти максимальної швидкості збору інформації, а отже і її оновлення для користувачів веб-застосунку.

Процес збору даних можна умовно розділити на три етапи. Перший етап полягає у зборі посилань на усі необхідні веб-сторінки сервісу, що є джерелом даних та збереження їх змісту (HTML розмітки сторінки). Другий етап полягає у фільтрації кожної отриманої сторінки за допомогою правил, що визначено розробником. Правила спроектовані таким чином, щоб залишити лише необхідну інформацію, та перетворити її у структури, які використовуються для зберігання інформації у сховищі даних. Третій етап полягає у збереженні отриманих даних, шляхом їх відправлення до віддаленого сховища даних. Для зв'язку зі сховищем даних розроблено окремий модуль у складі глобального модулю збору даних. Модуль зв'язку містить всередині себе необхідний функціонал для ініціації зв'язку та передачі інформації до сховища для збереження. Цей функціонал він надає (експортує) у вигляді набору функцій, що приймають дані у потрібному форматі. Після виконання усього процесу, або виникнення помилки, до модулю інтерфейсу командного рядка повертається сигнал, що містить інформацію про стан завершення операції. Він виводиться у вигляді інформації, що зрозуміла для людини (розробника чи адміністратора). При виникненні помилок, вся інформація про них зберігається у окремих текстових файлах, що дозволяє згодом проаналізувати причини виникнення помилок та усунути їх.

На першому етапи збору даних важливу роль відіграє функціональність, що вбудована у NodeJs та представлена модулем «crawler». Він і дозволяє отримати вміст сторінки, знаючи її URL адресу. Для кожного скрипту створюється об'єкт класу «CrawlerWrapper», що слугує обгорткою для об'єкту типу «crawler» із бібліотеки. Тип crawler реалізує шаблон «черги повідомлень», що використовується для потокової асинхронної обробки задач на збір даних. Також клас «CrawlerWrapper» має метод «enqueueTask», що додає кожну нову задачу до черги.

Кожна задача є об'єктом класу, що наслідується від розробленого базового класу «AbstractScrappingTask». Свій дочірній клас створюється у кожному скрипті, та представляє правила пошуку адреси та правила фільтрації даних, що

будуть отримані з веб-сторінки із заданою адресою.

Клас AbstractScrappingTask має наступні головні поля (деякі поля починаються із символу нижнього підкреслення «_», що є стандартною домовленістю, при розробці програм мовою JavaScript, для позначення поля як таке, що є приватним и не повинно використовуватися у зовнішньому коді за межами класу):

- uri. Зберігає URL адресу сторінки, з якої буде збиратися інформація у поточному об’єкті класу;
- _marketplace. Зберігає строковий ідентифікатор (назву) джерела даних. Він використовується для правильного сортування даних при перетворенні їх у структури, що зберігаються у сховищі даних;
- _sessionId. Зберігає числовий ідентифікатор, що означає порядковий номер поточного збору даних. Він використовується для правильного оновлення даних (щоб не затерти нові дані старими);
- _lang. Зберігає строковий ідентифікатор мови, якою збирається інформація. Це поле напряму зберігається у сховищі даних, разом із зібраною інформацію;
- asyncCallback. Зберігає об’єкт функції, що приймає зміст сторінки та виконує фільтрацію за правилами, що задані розробником скрипту. Правила також зберігається всередині даної функції. Після фільтрації вона перетворює дані у структури для зберігання у сховищі даних (функціонал для цього імпортується із модулю доступу до сховища). Повертає сигнал успіху збору та збереження даних, або помилку та інформацію про неї, якщо збір, фільтрація, або збереження пройшли невдало. Дана функція відмічена ключовим словом «async», що повідомлює інтерпретатору про необхідність оптимізувати її під асинхронне виконання;
- callback. Зберігає об’єкт функції, що викликається у момент, коли черга на обробку об’єктів збору даних доходить до поточного об’єкту. Дана

функція викликає асинхронно викликає функцію asyncCallback. При закінченні, посилає на сигнал про закінчення поточного етапу обробки;

– buildOptions. Зберігає об’єкт функції, що є виключно службовою та призначення для перетворення об’єкту класу AbstractScrappingTask на об’єкт класу, який очікує на вхід черга обробки об’єкту типу crawler;

Для спрощення фільтрації даних використовується бібліотека під назвою «x-ray». Вона дозволяю писати рекурсивні правила для пошуку даних у HTML розмітці на основі правил каскадних таблиць стилів (CSS).

Висновок до розділу 4

У даному розділі було спроектовано та частково реалізовано архітектуру веб-застосунку. Спроектовані структури даних для зберігання інформації у всіх структурних елементах застосунку.

Спроектовані окремі моделі даних для бази даних, для використання даних із сховища у коді застосунку та для зберігання інформації на клієнтському пристрої упродовж сеансу роботи. Розроблено діаграми розгортання (мовою UML) та схему базі даних, які були винесені до графічної документації проекту.

Також був спроектований та реалізований модуль для автоматизованого збору даних для веб-застосунку. Модуль був розроблений таким чином, щоб можливо було легко доповнювати список джерел даних новими сервісами.

5 КЛІЄНТСЬКИЙ ІНТЕРФЕЙС ВЕБ-ЗАСТОСУНКУ

Як уже згадувалося, основний механізм створення користувацького інтерфейсу, за допомогою веб-фреймворку VueJs, це компоненти. Компонентами у екосистемі VueJs називаються екземпляри класу Vue, що надає розробнику сам фреймворк. Вони об’єднуються у ієрархічне дерево, де кореневий компонент представляє увесь застосунок, а листові – логічно найменші елементи інтерфейсу (кнопки, поля вводу, тощо).

Компоненти складаються із трьох логічних частин. Першими двома є звичайна браузерна розмітка (HTML) та таблиця стилів до неї (CSS). Разом вони складають шаблон, який відповідає за відображення даних компоненту. Третьою частиною, яка є головною можливістю компонентів VueJs, є дані і логіка компоненту, які, з точки зору розробника, виглядають як поля та методи екземпляру компонента. Таким чином, архітектура компонента дуже нагадує шаблон MVVM. Модель представлена даними, що отримує і зберігає компонент. Представленням є шаблон компоненту. А моделлю представлення є логіка, що поміщена розробником до об’єкту компоненту. Вона відповідає за ін’єкцію даних у шаблон для їх коректного відображення, та інтерактивність компоненту. На схожість із шаблоном MVVM указують і розробники фреймворку. Так наприклад, екземпляри об’єктів компонентів у коді іменують vm (від ViewModel).

Згадана у розділі 3 реактивність у VueJs реалізується за рахунок неявного перетворення усіх даних у компонентах у пари гетер/сетер, та групування їх у ланцюжки викликів[15]. Це забезпечує миттєву послідовну зміну даних у всіх місцях застосунку де використовується одна і та сама змінна компоненту. А оскільки від даної змінної може залежати значення будь-якої іншої змінної, воно також оновиться до актуального стану. И так продовжуватиметься рекурсивно, доки не закінчиться виконання усіх ланцюжків сетерів.

Передача даних між компонентами реалізується по різному, у залежності

від напрямку поширення даних у ієрархічному дереві.

Від батьківських компонентів до дочірніх дані передаються у вигляді вхідних параметрів, що відповідають принципам реактивності. Для коду, що виконується всередині компоненту, вхідний параметр виглядає як звичайна змінна, проте зміна даних батьківського компоненту автоматично (без участі розробника) ініціює зміну даних у дочірньому компоненті, оновлюючи внутрішнє значення вхідного параметру до актуального.

Дані від дочірніх до батьківських компонентів передаються за допомогою подій, інструментарій для створення, ініціації та прослуховування яких надає фреймворк. Події реалізують класичний шаблон проектування «публікація-підписка». Ініціюючи зміну даних, дочірній компонент ініціює подію, яку можна прослуховувати у батьківському компоненті. Далі, батьківський компонент сам вирішує чи прослуховувати кожну подію, та як її обробляти.

Як можна бачити, реактивними є лише потоки розповсюдження даних униз по ієрархічному дереву. Таке обмеження гарантує цілісність та єдність даних у всіх частинах системи, за рахунок унеможливлення циклів та передбачуваності стану даних застосунку.

5.1 Загальна структура дерева компонентів

Як уже було згадано, увесь інтерфейс застосунку, що складається із компонентів є ієрархічним деревом. Усе відображення на клієнтський частинні веб-застосунку представлено одним кореневим компонентом, що має назву «App», від англійського слова «Application», що означає «застосунок». Його структура не має власної розмітки сторінки, а тільки містить три інших компоненти. Це компоненти «Header» та «Footer», що відповідають за верхню (шапка) та нижню (підвал) частини інтерфейсу. Вони є спільними для усіх сторінок застосунку і не змінюються при переходах за внутрішніми

посиланнями у веб-застосунку. Далі вони будуть іменуватися, відповідно, хедер та футер.

У хедері (рисунок 5.1) мітяться:

– логотип веб-застосунку, який є також посиланням на головну сторінку застосунку;

– форма, що містить текстове поле та кнопку, для пошуку товарів у каталозі;

– посилання на найпопулярніші серед усіх користувачів категорії товарів;

– іконки із підписом для списку «бажаного» та списків порівняння. При кліку на них, відкриваються відповідні випадаючі вікна;

– посилання на версії застосунку іншими мовами.

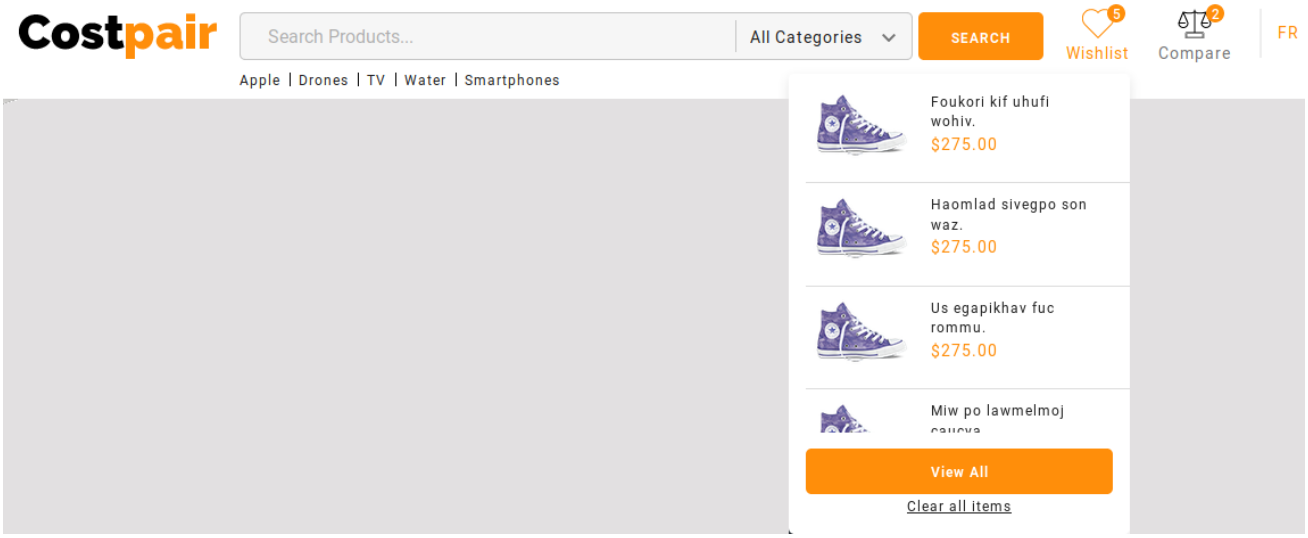

Рисунок 5.1 – Інтерфейс «шапки» веб-застосунку

У футері (рисунок 5.2) містяться:

– контактні дані для зворотнього зв’язку із компанією-власником сервісу (фізична адреса для листів, телефонний номер та адреса електронної пошти);

– посилання на службові документи сервісу, такі як правила користування, політика конфіденційності, розділ запитань-відповідей, правила доставки та повернення товарів, тощо);

– форма для вводу адреси електронної пошти для підписки на інформаційну розсилку від сервісу;

– коротка інформація про сервіс та копірайт.

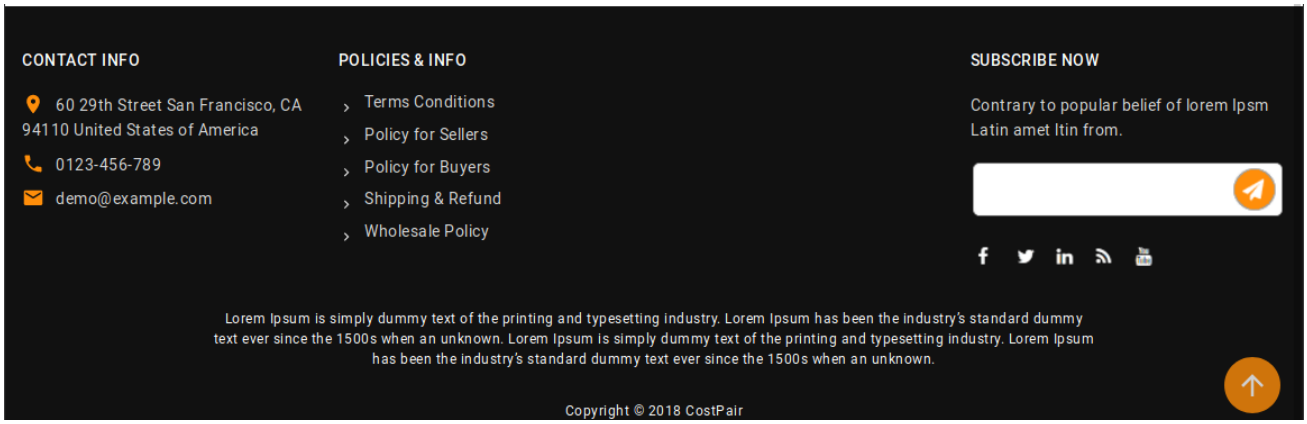

Рисунок 5.2 – Інтерфейс «підвалу» веб-застосунку

Кожна із представлених складових хедеру та футеру є окремим компонентом.

Третій дочірній компонент у компоненті App пов’язаний із системою маршрутизації та є унікальним для кожної сторінки застосунку.

5.2 Компоненти представлення

Кожна унікальна URL адреса застосунку, що прописана у конфігурації маршрутизації, пов’язана із певним великим компонентом, який називається «представлення». При запиті користувача, на певну URL адресу, система

маршрутизації визначає, який компонент відповідає даній адресі, та передає його на місце третього компоненту у корені компоненту App, між хедером та футером. А при користувацькому переході по внутрішнім посиланням у веб-застосунку, замість нового запиту на сервер, створюється новій компонент, що відповідає новій адресі, та замінює компонент минулої адреси. Таким чином реалізується функціональність односторінкового застосунку (SPA). Для користувача це непомітно, проте значно оптимізує швидкість роботи веб-застосунку.

Кожен компонент представлення має дочірні компоненти, тобто не є листом дерева компонентів. Він може автоматично отримувати, у якості вхідних параметрів, частини URL адреси, яка відповідає даному представленню. Наприклад, при переході за адресою типу «/category/11», у компонент представлення може бути передано вхідний параметр «categoryId», який буде дорівнювати «11». Варто зазначити, що тип даної змінної буде визначено як string, тобто рядок. Але, за допомогою засобів мови JavaScript (наприклад вбудовані функції «parseInt» або «parseFloat») можна перетворити рядок, що містить число, у змінну числового типу.

Усього застосунок містить 6 компонентів представлення:

- домашня сторінка (HomeView). Перша сторінка на яку потрапляє користувач. Має рекламне та ознайомче призначення. Містить посилання на популярні або рекламні товари та категорії;
- навігаційна сторінка категорії (CategoryView). Власна сторінка категорії, яка не є листовою у ієрархічному дереві. Містить опис категорії та посилання на підкатегорії;
- сторінка каталогу категорії (CategoryListView). Власна сторінка категорії, яка є листовою у ієрархічному дереві. Містить каталог товарів, та список фільтрів для більш точного пошуку товарів у каталозі;
- власна сторінка товару (ProductView). Містить детальну інформацію про

товар, список усіх продавців, які пропонують даний товар, із цінами та умовами доставки від продавців;

– сторінка порівняння товарів (CompareView). Містить таблицю, де у першому стовпці містяться назви характеристик товарів, а кожен інший стовпець представляє один із товарів, що порівнюються;

– сторінка перегляду списку «бажаного» (WishlistView). Містить список «бажаного» користувача, що відображається тими ж засобами, якими відображається каталог товарів у категорії.

Ще одна перевага базових компонентів полягає у тому, що вони слугують точками розділення коду на окремі набори. При старті веб-застосунку (перший запит користувача на URL адресу) формується великий файл (бандл), із готовими даними та розміткою, що прискорює час до інтерактивності, покращуючи враження користувача. Але передача усього дизайну компоненту однією відповіддю може занадто сповільнити час до першого байту цієї відповіді зі сторони користувача, що негативно відображається на враженні від веб-застосунку. Асинхронне завантаження компонентів представлення вирішає дану проблему. При першому запиті користувач отримує у відповідь лише один компонент представлення, що йому необхідний. А при наступних запитах не потрібно заново передавати код для формування хедеру чи футеру, а лише новий компонент представлення. Більш того, ці компоненти можуть завантажуватись у фоновому режимі, після першої відповіді. Це, непомітно для користувача, передасть усю необхідну інформацію, забезпечивши надалі максимальну швидкодію веб-застосунку.

5.3 Базові компоненти

Зазвичай, для використання одного компоненту всередині іншого, дочірній компонент треба явно імпортувати на реєструвати у батьківському. Такий підхід

реалізовано із метою оптимізації, щоб не зберігати у пам’яті усі компоненти, навіть якщо вони не використовуються. Але, інколи, деякі є зовсім маленькими, а використовуються часто и майже на кожній сторінці. Зазвичай це є базові компоненти управління користувацького інтерфейсу.

Для таких компонентів є можливість глобальної реєстрації. Вона проходить при запуску веб-застосунку. Глобально зареєстровані компоненти не потребують явного імпорту і можуть використовуватися у будь-якому місці коду застосунку. Недолік даного підходу полягає у тому, що ці компоненти будуть завжди знаходитися у пам’яті, незалежно від того, потрібні вони чи ні. Тому глобально рекомендується реєструвати лише зовсім маленькі компоненти, що дуже часто використовуються.

У даному веб-застосунку, прикладами базових компонентів є:

- кнопки;
- два варіанти прапорців для різних функцій інтерфейсу;
- компоненти, що відповідають за відображення іконок в інтерфейсі веб-застосунку;
- випадаючий список;
- каталог для попереднього перегляду товарів.
- пагінація для каталогу товарів;
- компонент, що реалізує функціонал випадаючих вікон (частин інтерфейсу);
- два компоненти, що реалізують функціонал вкладок (табів). Вони невід’ємно пов’язані між собою. Перший відповідає за загальну функціональність, а другий представляє собою кожну окрему вкладку.

Базовими компонентами представлено увесь набір мінімальних елементів управління для користувача.

5.4 Кольорова гама

Вибір кольорів для використання у дизайні є однією із найважливіших частин створення дизайну користувацького інтерфейсу. Від емоційного відтинку кольорів та їх поєднання між собою. Два спрощення розробки дизайну було прийнято рішення використовувати два основні кольори.

Для гарного взаємного контрасту пари кольорів, вони повинні бути обраними із протилежних частин кольорового спектру. За даними дослідження Йоганенеса Іттена, сірий колір може бути доданий у будь-яку пару кольорів[16]. Це зумовлюється різноманіттям його відтінків, що можуть бути будь якої яскравості на насиченості, від чорного до білого. У контрастний набір до сірого кольору брати лише один яскравий.

Другим кольором було обрано помаранчевий. Він суб’єктивно асоціюється у користувачів із бадьорістю та активністю, що буде спонукати їх більш активно користуватися сервісом. Увесь помаранчевій колір, що представлено у дизайні єдиним відтінком, щоб складати враження впевненості.

Як додаткові фонові кольорі використовуються чорний та білий, що складають класичну пару та доповнюють використання сірого кольору.

Для виділення важливих функцій, у поодиноких випадках, додаються синій та зелений кольори що розташовуються виключно подалі від помаранчевого на білому фоні.

Висновок до розділу 5

У даному розділі було переведено увагу з архітектурних рішень до рішень проектування та реалізації зовнішнього вигляду користувацької частини веб-

застосунку. Було спроектовано ієрархічне дерево компонентів, що дозволило реалізовувати кожен елемент користувацького інтерфейсу окремо. Таке послаблення зв’язності полегшує проектування коду та його повторне використання.

Оптимізації передачі даних, наведені у цьому розділі, дозволили покращити швидкодію, гармонічно об'єднавшись із архітектурою, що була спроектована у попередньому розділі.

Приділення уваги до кольорів та ергономіки повинно покращити враження користувачів від веб-застосунку та спонукати їх активно користуватися сервісом.

ВИСНОВКИ

Даний проект мав на меті розробку сучасного та функціонального веб-застосунку на актуальну тематику. У даному проекті було спроектовано та розроблено веб-застосунок, що має функціонал автоматизованого збору даних від найрізноманітніших продавців товарів у мережі Інтернет. Користувачеві даний застосунок дає змогу зручно переглядати, шукати, порівнювати та зберігати будь-яку із зібраної інформації. Все це працює у межах одного веб-застосунку.

Як показав огляд предметної області, таке поєднання інформації із різних куточків мережі Інтернет дійсно є актуальною потреб користувачів «всесвітньої павутини». На основі цього огляду, а також аналізу достатньої кількості веб-сервісів схожої тематики були сформовані чіткі основні вимоги до функціональних можливостей, архітектури та користувацького інтерфейсу веб-застосунку, що розроблено у даному проекті. Усі вимоги були точно дотримані.

Розроблений веб-застосунок має порівняно невеликий набір функцій, але усі вони є актуальними, та часто використовуються користувачами. Відмова від реалізації функціоналу, що використовується не так часто (наприклад відсутність «власного кабінету» користувача) дозволила не перевантажувати дизайн застосунку, зробивши його зрозумілим та зручним для користувачів.

Користувацький інтерфейс веб-застосунку був розроблений із урахуванням актуальних трендів, було обґрунтовано підібрано невелику, але контрастну гаму кольорів, для забезпечення приємного психологічного враження від користування застосунком.

Модуль для збору даних було реалізовано таким чином, щоб його можна було легко розвивати, додаючи нові джерела даних. Також всі дії у ньому легко поєднуються у необхідні набори та автоматизуються за рахунок реалізації повнофункціонального інтерфейсу командного рядка. Після збору інформація

відразу попадає у сховищі даних, що дає користувачам можливість завжди бачити найактуальнішу інформацію, що їх цікавить.

Для розробки проекту було обрано набір (стек) технологій для розробки програмного забезпечення, що легко поєднуються між собою та забезпечують швидкість роботи та можливості для легкого розширення функціоналу розробленого застосунку. Важливим критерієм вибору технологій була відкритість коду та ліцензій на дані технології. Абсолютно усі використані технології є вільним та відкритим програмним забезпеченням, що дає більшу впевненість у аспекті інформаційної безпеки веб-застосунку, бо популярне відкрите програмне забезпечення постійно перевіряється великою спільнотою розробників, що його використовують. Крім того, важливим аспектом даного вибору є те, що використання даного програмного забезпечення безкоштовно і безпечно із юридичної точки зору. Це важливо, так як матеріали, що розроблені у рамках дипломного проекту, представляють інтерес для зарубіжних компаній, що працюють у сфері електронної комерції. Безкоштовність використаного програмного забезпечення дозволило розробити великий веб-застосунок майже без фінансових затрат.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. eMarketer Worldwide Retail and Ecommerce Sales: eMarketer's Updated Forecast and New Mcommerce Estimates for 2016—2021. Executive Simmary. 29.01.2018. [Електронний ресурс] : Режим доступу: https://www.emarketer.com/Report/Worldwide-Retail-Ecommerce-Sales-eMarketers-Updated-Forecast-New-Mcommerce-Estimates-20162021/2002182.
2. Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides Design Patterns: Elements of Reusable Object-Oriented Software. AddisonWesley Professional, 10.10.1994. 416 с.
3. TypeScript. JavaScript that scales [Електронний ресурс] : Режим доступу: https://www.typescriptlang.org/ – Назва з екрану.
4. Brett McLaughlin From Web sites to Web applications, Part 1: Web site or Web app? 01.06.2010. [Електронний ресурс] : Режим доступу: https://www.ibm.com/developerworks/web/library/wa-websiteapp/wa-websiteapp-pdf.pdf.
5. Flux. Application architecture for building user interfaces [Електронний ресурс] : Режим доступу: https://facebook.github.io/flux/ – Назва з екрану.
6. Jason Miller, Addy Osmani Rendering on the Web. 29.05.2019. [Електронний ресурс] : Режим доступу: https://developers.google.com/web/updates/2019/02/rendering-on-the-web.
7. Vue SSR Guide Client Side Hydration. [Електронний ресурс] : Режим доступу: https://ssr.vuejs.org/guide/hydration.html.
8. Webpack Why webpck. [Електронний ресурс] : Режим доступу: https://webpack.js.org/concepts/why-webpack.
9. NodeJs About Node.js. [Електронний ресурс] : Режим доступу: https://nodejs.org/en/about/.

10. Azat Mardan Master Express.js: The Node.js Framework For Your Web Development. Apress, 2014. 372 с.
11. Alex Kriegel Discovering SQL: A Hands-On Guide for Beginners. Wrox, April 2011. 499 с.
12. Mohamed Aladin Software Architecture - The Difference Between Architecture and Design. 27.07.2018. [Електронний ресурс] : Режим доступу: https://codeburst.io/software-architecture-the-difference-between-architecture-and-design-7936abdd5830.
13. Vijini Mallawaarachchi 10 Common Software Architectural Patterns in a nutshell. 04.09.2017. [Електронний ресурс] : Режим доступу: https://towardsdatascience.com/10-common-software-architectural-patterns-in-a-nutshell-a0b47a1e9013.
14. Мирошниченко Г. А. Реляционные базы данных: практические приёмы оптимальных решений. Санкт-Петербург : БХВ-Петербург, 2005. 400 с.
15. Hassan Djirdeh, Nate Murray, Ari Lerner Fullstack Vue: The Complete Guide to Vue.js. 1 edition. CreateSpace Independent Publishing Platform, 01.04.2018. 442 с.
16. Иоханнес Иттен Искусство цвета: [пер. с нем.]. Москва : Д. Аронов, 2004. 95с.

ДОДАТОК А

Діаграма класів моделі коду веб-застосунку мовою UML